
programación
avanzada para
el zx spectrum
LeG
indescomp

Programación Avanzada para el ZX Spectrum

indescomp



SHIVA PUBLISHING LIMITED
4 Church Lane, Nantwich, Cheschire CW5 5RQ. England

Reprinted 1983 (Twice)

ISBN 0 906812 24 0 (Primera edición)

Versión en castellano

Editado por **INDESCOMP, S.A.**
Pº de la Castellana, 179 - 28016 Madrid

*Todas las consultas relativas a este libro deberán ser dirigidas a **INDESCOMP, S.A.**

Traducción, adaptación e impresión **CONORG, S.A.**

I.S.B.N.: 84-86176-08-5

Depósito Legal: M-14591-1984



Contenido

	Prefacio	1
1	Mapamundi	3
2	Rellenando bloques	6
3	Funciones definidas por el usuario	17
4	Caracteres de Control	22
5	Técnicas de imagen	28
6	Variables sistemales	35
7	Fichero de atributos y grafismos	46
8	Psicoespectrología	51
9	Ficheros	57
10	La estadística en forma simple	77
11	Mejorando la imagen	86
12	Renumerando líneas	90
13	Polígonos	96
14	Criptografía y Criptoanálisis	101
15	Cambiando el Repertorio de Símbolos	107
16	Trazado de Curvas sin encontronazos	112
17	Sistemas de Gestión de Archivos	125
18	Las Cartas Astrales	143

Apéndice A: El Sistema de Archivos en Cassettes - Referencia y Descripción del sistema 153

Apéndice B: Control Automático del cassette 160

Apéndice C: Una Guía del Usuario para el SDM - El Manager de Datos Spectrum 161

Apéndice D: El Manager de Datos Spectrum - Listado del Programa 165

Apéndice E: Construyéndote tu conmutador Cargue/Guarde 170



Prefacio

Tienes un Sinclair ZX Spectrum y te sientes con bastante confianza sobre cómo usarlo. Sabes lo que las teclas hacen, y puedes ensartar veinte o treinta líneas de BASIC y hacer que funcione. Has repasado concienzudamente el Manual y un libro de introducción. Has tecleado docenas de programas sacados de las revistas y descubierto que los programas breves hacen todos lo mismo, y que los largos, si no están llenos de errores, te llevan horas y horas de trabajo cuidadoso -y meramente por ochocientas pesetas puedes comprar un cassette que produce resultados más impresionantes. Todo eso estaría bien excepto que no quieres continuar gastando ochocientas pesetas para comprar los programas de otra gente- lo que quieres es producir tus propios programas.

Así que, ¿ahora qué?

Todavía te queda bastante camino que recorrer antes de que puedas escribir juegos en Código Máquina con calidad competitiva con los de paredes de ladrillos, o programas que muestren el firmamento, tal y como estaba en cualquier época entre el año 4000 a.d.C. o estará en el 6000 d.d.C.; y mientras que este libro puede iniciarte a lo largo de ese camino, no te llevará ciertamente durante todo él.

Lo que **sí** hará es ayudarte a ampliar ambas capacidades, la tuya y la de tu máquina.

Hay tres direcciones principales para explorar.

Una pudiera describirse como "Teoría de Computación": cómo desarrollar técnicas para mejorar tus programas. Por lo que respecta a este libro, hemos adoptado una visión bastante práctica de lo que constituye la teoría: es decir, nos hemos concentrado sobre rasgos específicos del Spectrum, tales como sus posibilidades de color y sus gráficos, y hemos indagado un poquito más sobre la máquina. Encontrarás más cosas sobre caracteres de control, funciones definidas por el usuario, gráficos definidos por el usuario, variables del sistema, y los ficheros de atributos y grafismos, y sobre cómo sacar mayor provecho de ellos.

La segunda dirección es "Mejora de la Máquina". Escribiendo adecuados utensilios de programación y explotación, puedes equipar a tu Spectrum con facilidades que la máquina desnuda no posee. La rápida renumeración de líneas en los programas en BASIC (nuestra rutina te permite entresacar un bloque de un programa y renumerar ese bloque por sí mismo -lo que es bastante bueno para perfeccionar subrutinas). Trazando gráficos sin tener que preocuparse de si los puntos se salen de la pantalla. El relleno automático de bloques para dibujos a líneas. Sistemas efectivos para manejar grandes cantidades de información conservadas en las cintas del cassette como **ficheros**, que pudiera usarse como base de un sistema práctico de registro de discos, para el hogar o para la industria. De forma gradual, te llevaremos desde un simple sistema de archivos en cassette a un sistema de gestión de datos.



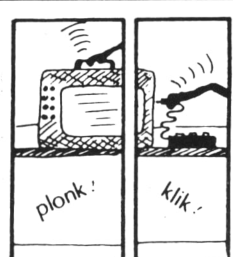
En tercer lugar... bien, hemos notado que siempre que preguntas a un entusiasta de los computadores: "muy hermoso, pero ¿qué puedo hacer yo con ésto?" tiende a cambiar de tema. Es como si el objetivo primordial de la computación, fuera hacer **más** cómputo. Meramente el Arte por el Arte, los ordenadores como "estilo de vida". Pero ¿no sería hermoso usar realmente el computador para hacer algo más? Aquí encontrarás algunas sugerencias: mapas, posiciones de los astros, experimentos psicológicos, estadística simple, criptografía y criptoanálisis, manipulación de símbolos.

Dos áreas que aquí no tratamos son Código Máquina y teoría "pura" -temas como estructura de datos y programación estructurada. En otros libros tratamos sobre ellos, en Código Máquina y Mejor Basic y en Código Máquina del Spectrum.

Nuestro objetivo primario no es producir programas altamente pulidos "dispuestos para el horno". El énfasis principal está en el proceso doloroso pero satisfactorio, de desarrollar una idea inicial en un programa que **funciona.** En lugar de presentar simplemente el resultado final, describimos algunas veces las rutinas que luego son modificadas, vueltas a escribir, revisadas o desechadas completamente y sustituidas por otras. Después de todo, así es como cualquier programa que no sea trivial acaba siendo escrito, y lleva a confusión pretender lo contrario. No intentamos darte la impresión de que escribir programas pueda o deba ser fácil y sin esfuerzo. El punto importante es que **todo el mundo** comete errores, así que no hay razón para desanimarse cuando tú los hagas. El truco es reconocer los errores y hacer que dejen de serlo. Desde luego, cualquier método que ayude a reducir las posibilidades de producir errores es digno de tenerse.

Adicionalmente, sin embargo, algunas de las rutinas utilitarias más generalmente aplicables, están listadas separadamente en los apéndices, de manera que no es necesario emprenderlas con las descripciones de cómo se construyen, para ser capaz de usarlas. Si todo lo que quieres es copiar los listados y pasar el programa, puedes hacerlo. Como en todos nuestros libros, hemos intentado mantener las explicaciones claras y simples. Este libro no es un curso rígidamente estructurado: está diseñado para que tú lo vayas ojeando y estudiando a tu gusto. Algunos capítulos sí dependen de los anteriores de alguna manera, pero es siempre obvio cuando así ocurre. Así que, comienza por poner el dedo en aquellos capítulos que tengan un atractivo particular para tus gustos, y da una pasada sobre ellos primeramente. Los encontrarás muy instructivos. Y divertidos también.

Hasta ahora, nos hemos referido a nosotros mismos (los autores) como "nosotros" -pero como en el libro Fácil Programación **hemos encontrado que no siempre funciona más adelante. Así que, a partir de ahora, nos referiremos a nosotros mismos, a pesar de ser dos, en singular y como "yo". Y siempre que digamos "nosotros", queremos decir "yo, el autor y tú, el lector". Puede parecer una idea absurda, pero realmente resulta más informativo de esa manera.**







Cuanto mayor sea el esfuerzo que estés dispuesto a desarrollar, más entrenado estará tu Spectrum para realizarlo. Gasta un par de horas, y podrás tener un...

I Mapamundi

Es una posibilidad. La misma técnica te permite dibujar y guardar en cassette, dibujos a línea en alta resolución, tales como caricaturas de Isaac Newton u Olivia Newton-John, o del paisaje marciano para que lo uses en programas con Trápalas del Espacio.

Es fácil; pero exige tiempo. Aquí tienes un diagrama de un mapa-mundi en Spectrum, simplemente para convencerte de que los resultados justifican ampliamente el tiempo empleado.

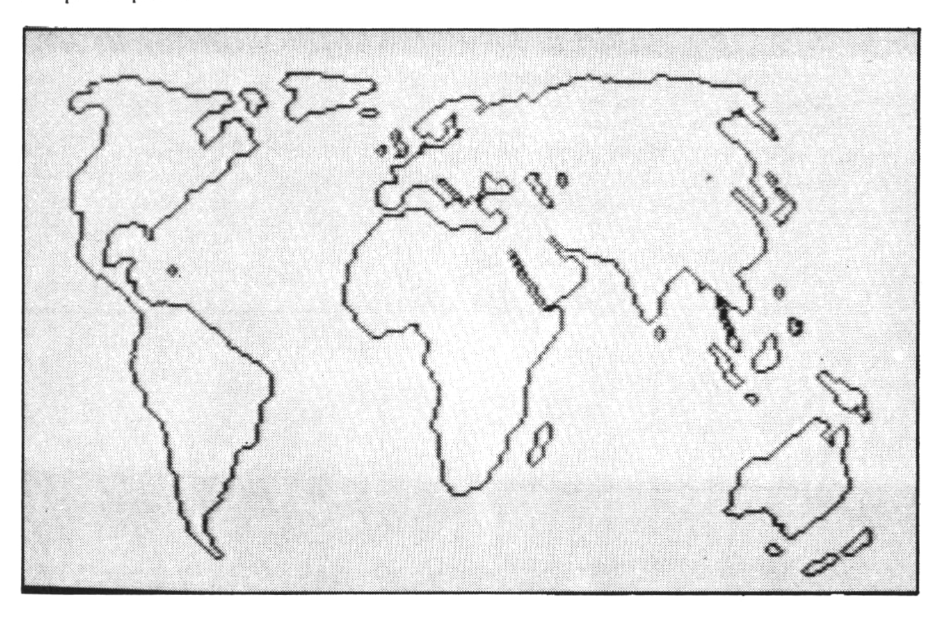


Figura 1.1 Un mapa de contornos del mundo, producido en un Spectrum.

La forma difícil de meter esto en la máquina, es enfrascarse sobre las retículas de latitud y longitud, copiando las coordenadas, y abasteciendo con ellas una rutina de dibujo.

La forma en que yo realmente lo he hecho, es basta pero efectiva.

1. Corta una lámina de plástico transparente -tal como parte de una bolsa de polietileno- según el tamaño de la pantalla de tu televisión.
2. Marca sobre ella el contorno de la zona central de la imagen en el Spectrum -la parte en la que puedes PRINT o PLOT. La forma fácil de hacerlo es teclear simplemente BORDER Ø.



3. Busca un mapa del mundo, justo del tamaño adecuado a este área.
4. Trázalo sobre el plástico, utilizando un rotulador de punta de fieltro. Saldrá una imagen bastante tosca.
5. Deja que se seque la tinta, y teniendo cuidado de no frotarla, pega el mapa sobre el frente de la pantalla, alineándolo con el área central, y usando cinta adhesiva.

Esos son los requisitos de "hardware" para este método. Vamos ahora con el "software".

DIBUJERO

6. Teclea un programa Dibujero, que te permita controlar y mover la "pluma" sobre la pantalla, por medio del teclado, de manera que pinte una mota, o bien se mueva sin pintar. También es digno de tener algún medio de borrar los fallos. Aquí hay un programa que hará ese trabajo: desde luego que puedes hacerlo más complejo si te sientes con ganas.

```
 10 LET x=0: LET y=175
 20 OVER 1
 30 LET estela=0
 40 INPUT d$
 50 LET x0=x: LET y0=y
 60 IF CODE d$<60 THEN  GO TO 100
 70 IF d$='m' THEN  LET estela=0
 80 IF d$='p' THEN  LET estela=1
 90 GO TO 40
100 REM Segun numero pulsado
110 GO SUB 10*CODE d$-290
120 IF estela=0 THEN  PLOT x0,y0
130 PLOT x,y: GO TO 40
200 LET x=x+1: LET y=y+1: RETURN
210 LET x=x+1: LET y=y-1: RETURN
220 LET x=x-1: LET y=y-1: RETURN
230 LET x=x-1: LET y=y+1: RETURN
240 LET x=x-1: RETURN
250 LET y=y-1: RETURN
260 LET y=y+1: RETURN
270 LET x=x+1: RETURN
```

7. Usando los controles del teclado (como explicamos en detalle más adelante) mueve la pluma hasta un punto situado por debajo de los contornos trazados, y sigue los contornos pintando la mota a medida que vas pasando, construyendo así el contorno de los continentes. Cuando hayas terminado de recorrer un continente, pasa sin apoyar la pluma hasta el siguiente, y luego comienza de nuevo a dibujar.

Lo ves, realmente es fácil. Y además los resultados pueden ser magníficos si le echas tiempo y cuidado para no equivocarte.

USANDO EL DIBUJERO

El programa puede estar en uno de dos "modos":

m: MUEVE la pluma a una nueva posición;
p: PINTA la posición corriente a medida que se mueve la pluma.



Se comienza con "m". En cualquier momento, puedes cambiar los modos tecleando el símbolo "p" o el "m".

El movimiento está controlado por las teclas 1-8 como sigue:

4	7	1
5	*	8
3	6	2

La pluma se mueve a partir de su posición corriente (*) una mota hacia arriba, hacia abajo, lateralmente o diagonalmente según los números. (El orden puede parecer extraño: la idea es que las teclas "flecha" 5-8 funcionen como habitualmente, y los movimientos "diagonales" 1-4 comiencen como el reloj a la 1 y se muevan a derechas).

El programa, tal y como está, exige que teclees cada número y el símbolo del modo (pulses dos teclas). Puedes usar, si lo prefieres INKEY$; pero de esta forma, tienes la posibilidad de comprobar que has pulsado la tecla correcta antes de que hagas algo equivocado.

Experimenta con los movimientos. Dado que usamos la instrucción OVER 1, si pintas dos veces en el mismo sitio, la mota correspondiente se queda en blanco. Eso te permite borrar los errores. Sin embargo, debes llevar en cuenta dos cosas:

1. Para pasar a una nueva región, haz un desplazamiento **alejándote** de tu curva acabada (en una dirección que luego no tropiece con ella) **antes** de cambiar a modo "m".
2. Al llegar a un nuevo trozo de curva, no pulses "p" hasta que tu pluma esté exactamente alineada con ella.

Si cometes errores al escoger los modos, la tendencia es que obtengas motas aisladas sobre la pantalla. Para quitártelas de encima, ponte en modo "m". Mueve la pluma hasta que estés encima de ellas (lo que las machacará); pasa al modo "p"; desplaza una posición; regresa al modo "m". Ensáyalo.

No esperes un resultado muy pulido en los primeros diez minutos.

GUARDANDOLO

Una vez que estés satisfecho, guarda en cinta el mapa trazado: no necesitas gastar otras dos horas pegado a la pantalla otra vez. Simplemente detén el programa Dibujero; luego teclea como un comando directo:

SAVE "mapita" SCREEN$

Para CARGARLO de nuevo en memoria, lo efectuarás mediante la rutina usual. Teclea LOAD "mapita" SCREEN$

Algunos de los capítulos posteriores de este libro, suponen que has dibujado un mapa (puede ser mucho más simple que el de mi foto) y que lo has guardado en cinta así que ¡A currar se ha dicho!











Algunas veces, el principal problema al escribir un programa es simplemente decidir qué es lo que la máquina tiene que hacer... como es el caso de este programa utensilio gráfico que sombrea regiones de la pantalla -con unas cuantas restricciones sobre el contorno de la figura.

2 Rellenando Bloques

La idea primitiva fue:"¿no sería bonito mostrar en pantalla un mapa-mundi en que las áreas concernientes a los mares estuvieran **ennegrecidas**?" Y el pensamiento inmediato fue "usando el programa Dibujero, me llevaría semanas". Desde luego que la idea fue conseguir que el Spectrum hiciera todo el trabajo. Suena fácil a primera vista, y en los casos simples lo es, pero no puedes decirle al Spectrum "encuentra las curvas cerradas y rellena la parte de dentro", porque el pobre no sabe lo que es una curva cerrada, ni tiene ningún Conocimiento Instintivo. Ni "desde luego" parece que este enfoque pueda funcionar a nivel de cómputo.

Comencemos con un caso fácil y desarrollaremos el mapa en etapas sucesivas. La tarea más simple es rellenar una región simple y cerrada, equivalente a un círculo o a un polígono. El título que buscamos es:

COLORERO

Supongamos que tenemos un polígono cerrado pintado en la pantalla. Ignoremos la pragmática por el momento, y consideremos la teorética: "¿Cuáles son los pasos que el ordenador debe seguir con el fin de rellenar la parte interna de un contorno?"

La respuesta es fácil:

1. Para cada línea horizontal, encontrar el punto extremo-izquierda del polígono a partir de la izquierda.
2. Encontrar el extremo-derecha del polígono buscando a partir de la derecha.
3. Unir ambos extremos mediante una línea horizontal.
4. Repetir para cada línea.



La forma de ver si una mota está pintada en la pantalla, es usar la función POINT (Programación Fácil*, página 36). El valor de POINT (x, y) es 1 si hay una mota trazada en (x, y); ∅ si no lo está. Así que el programa que queremos es éste:

```
 10 FOR y=0 TO 175
 20 LET x1=0
 30 IF POINT (x1,y)=1 THEN  GO TO 60
 40 LET x1=x1+1
 50 IF x1<=255 THEN  GO TO 30
 60 LET xr=255
 70 IF POINT (xr,y)=1 THEN  GO TO 100
 80 LET xr=xr-1
 90 IF xr>=0 THEN  GO TO 70
100 IF x1>=256 THEN  GO TO 120
110 PLOT x1,y: DRAW xr-x1,0
120 NEXT y
```

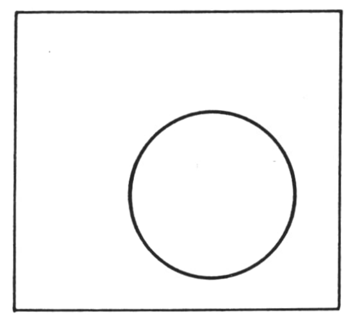
Como prueba, mete este programa; luego teclea el comando directo:

CIRCLE 127, 87, 87

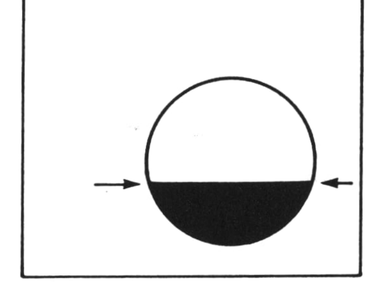
(por ejemplo) y a continuación teclea

GO TO 1∅

(**no** RUN, que borraría la pantalla!)



ANTES



DURANTE

Figura 2.1 Con figuras fáciles, sombreando desde el punto extremo-izquierda hasta el punto extremo-derecha funciona.

Es lento, estáte seguro (cualquier rutina que coloree lo va a ser; la cantidad de cómputo está predestinada a ser grande porque hay 45.056 motas en la pantalla de las que preocuparse) pero funciona.

Si la combinas con la rutina del capítulo 13 que dibuja polígonos, de manera que primero dibujes un polígono sencillo, luego lo colorees, verás que continúa funcionando.

* Programación Fácil del ZX Spectrum por Ian Stewart y Robin Jones, Shiva.



LAS PEJIGUERAS

Sin embargo, falla estrepitosamente cuando son varias las regiones que necesitan colorearse (y desde luego, también en otros casos). Ensaya:

```
CIRCLE 5Ø, 5Ø, 49: CIRCLE 16Ø, 5Ø, 49
GO TO 1Ø
```

Eso no era lo que pretendíamos, ¿verdad?.

Lo que está haciendo es encontrar el punto extremo-izquierda de un círculo, y luego el púnto extremo-derecha del **otro,** y uniendo esos dos. Una forma de arreglarlo es conseguir saber qué curva cerrada es cada una, pero eso parece bastante lioso y largo. Y, en todo caso, todavía tenemos problemas incluso aunque sólo haya una curva. Ensaya con esto:

```
PLOT 1ØØ, 5Ø: DRAW 5Ø, -5Ø: DRAW -5Ø, 1ØØ:
DRAW -5Ø, -1ØØ: DRAW 5Ø, 5Ø:
GO TO 1Ø
```

Es un polígono simplemente conexo (que dicen los matemáticos), con forma parecida a la de la punta de una flecha; y el programa nos colorea demasiado.

Aquí la razón es que ciertas líneas horizontales cruzan la figura en más de dos puntos. Por ejemplo, una línea como la mostrada en la figura 2.2, tropieza en cuatro puntos y solamente queremos colorear entre el 1º y el 2º y el 3º y el 4º. Pero **no** entre el 2º y el 3º.

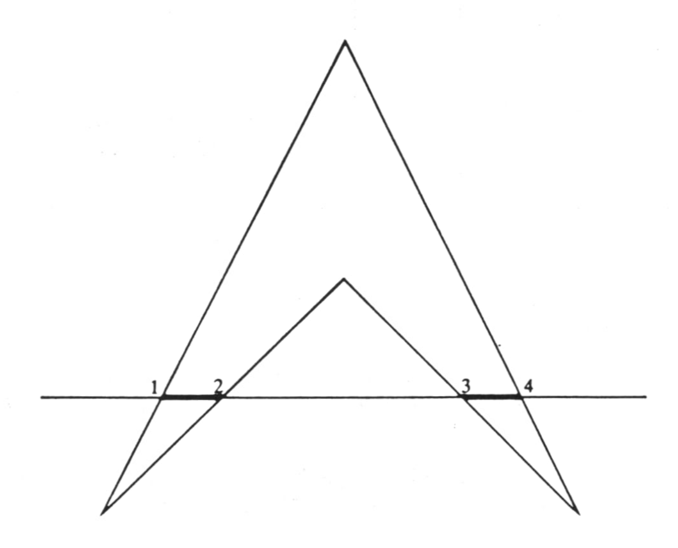


Figura 2.2 Sobre figuras más complicadas, no funcionará!



Lo que nos sugiere es ésto:

1. Rastrear a lo largo de la línea, buscando los puntos de cruce con la curva, y enumerarlos todos.
2. Dibujar del 1º al 2º, del 3º al 4º, ...y en general desde los impares (2 * i + 1) hasta los pares (2 * i + 2) variando la i desde 1 a lo que corresponda.

Si piensas que ahí está el truco, intenta escribir un programa para resolverlo. Luego comprueba ese programa con la figura de flecha citada.

Seguro que has dado un alarido.

Ya va mal en la primera línea. Sólo hay dos puntos; pero no quieres tirar una línea entre ellos, porque es la punta de un trozo de polígono que resalta, y entre ellos lo que hay es "una bahía".

Si pintas sobre pedazos de papel, llegarás a convencerte por tí mismo, que aparte de este problema de "las puntas de los cuernos" la idea puede funcionar. Por ejemplo, con las formas mostradas en la figura 2.3, colorea correctamente en las líneas A y B, pero no en la C ni en la D que tropieza con puntas. Nota que funciona incluso cuando hay varias curvas cerradas en el diagrama.

Así que el problema ahora es: "Cómo reconocer la punta del cuerno"

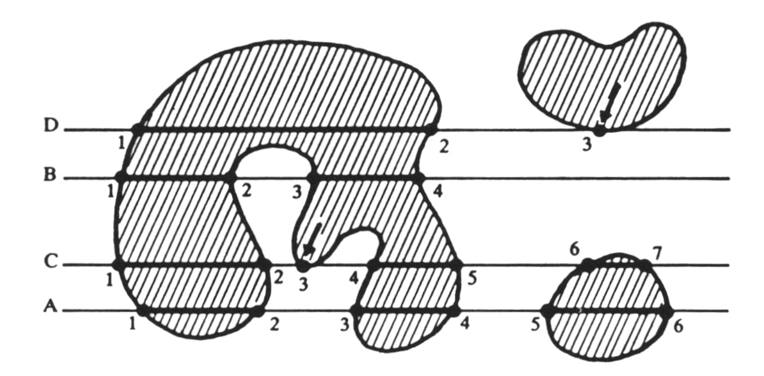


Figura 2.3 Rellenando entre las intersecciones impares y pares, sería el truco, excepto en los cabos de las penínsulas, como sucede en las líneas C y D.

Obviamente, el rasgo característico de un cabo es que la curva no **cruza** realmente la línea horizontal. Entra por un lado, posiblemente coincide con ella durante un rato; pero luego, la deja por el mismo lado en que entró.

Compara los dos casos mostrados en la figura 2.4.



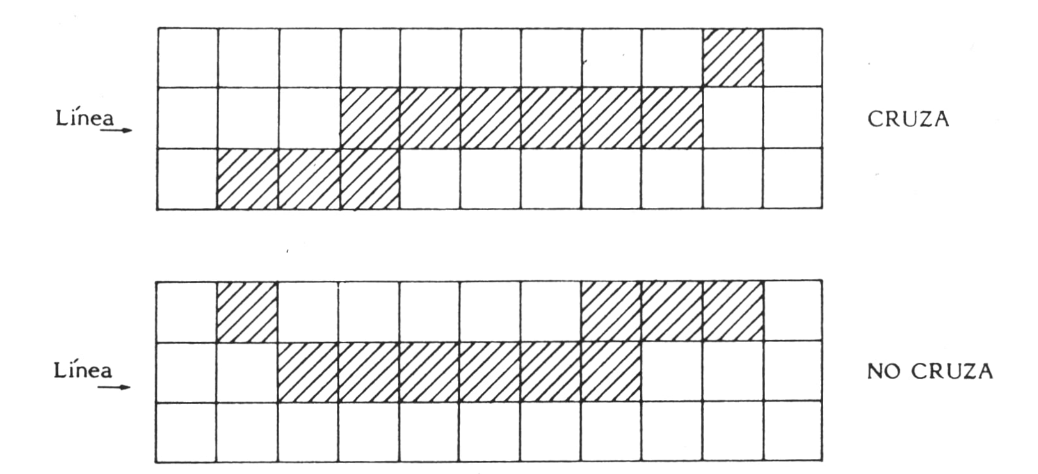


Figura 2.4 Para localizar los cabos de las penínsulas, se mira si la curva cruza la línea horizontal o no.

Así que lo que parece que necesitamos es una rutina que nos diga si la curva cruza la línea o no; y si lo hace, la ignoramos. Con el fin de ver si cruza, necesitamos también ser capaces de reconocer el trozo que circula **a lo largo** de la línea en que estamos interesados. Así que observaremos los cuadritos (pixels) que rodean los dos extremos de ese tramo, y miraremos a ver si están adecuadamente rellenos (mira figura 2.5).

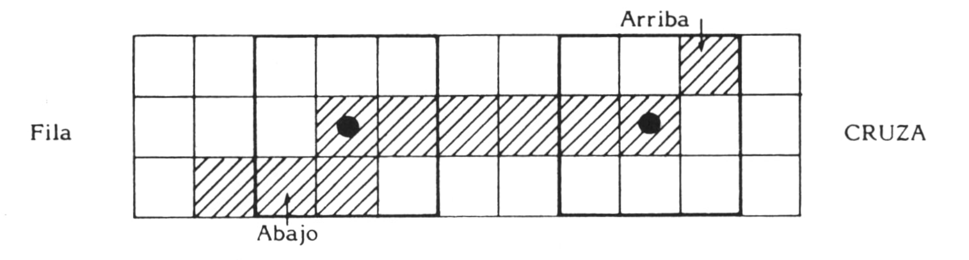


Figura 2.5 Detección de cruces, escrutando dos cuadrados de 3 x 3 cuadritos (pixels).

De hecho, todavía somos un poquito cándidos; pero hemos conseguido una idea desarrollable, y ella nos lleva al siguiente programa.

SUPERCOLORERO

```
 10 LET rastrea=1000
 20 LET prueba=2000
 30 LET lista=3000
 40 LET sombrea=4000
 50 DIM a(20)
 60 DIM b(20)
200 FOR y=1 TO 174
205 LET q=0
```




```
 210 LET x=1
 220 IF x>=255 THEN  GO TO 400
 230 IF POINT (x,y)=0 THEN  LET x=x+1: GO TO 220
 240 LET xl=x: GO SUB rastrea
 250 GO SUB prueba
 260 LET x=xr+1: GO TO 220
 400 GO SUB sombrea
 500 NEXT y
1000 REM rastrea
1010 LET c=0
1020 IF xl+c>=255 THEN  RETURN
1030 IF POINT (xl+c,y)=0 THEN  GO TO 1060
1040 LET c=c+1: GO TO 1020
1060 LET xr=xl+c-1
1070 RETURN
2000 REM prueba
2005 LET ll=0: LET lu=0: LET rl=0: LET ru=0
2010 FOR e=-1 TO 1
2020 IF POINT (xl+e,y-1)=1 THEN  LET ll=1
2030 IF POINT (xl+e,y+1)=1 THEN  LET lu=1
2040 IF POINT (xr+e,y-1)=1 THEN  LET rl=1
2050 IF POINT (xr+e,y+1)=1 THEN  LET ru=1
2060 NEXT e
2070 IF ll+rl=0 OR lu+ru=0 THEN  RETURN
2080 GO SUB lista
2090 RETURN
3000 REM lista
3010 LET q=q+1: LET a(q)=xr
3020 RETURN
4000 REM sombrea
4010 IF y<=1 THEN  RETURN
4020 FOR t=1 TO 20 STEP 2
4030 IF b(t+1)=0 THEN  GO TO 4100
4040 PLOT b(t),y-1: DRAW b(t+1)-b(t),0
4050 NEXT t
4100 FOR t=1 TO 20
4110 LET b(t)=a(t)
4120 NEXT t
4130 DIM a(20)
4140 RETURN
```

Antes de describir algunas de las peculiaridades de esta rutina, te sugiero que la ensayes primeramente. Tecléala, y luego añade:

```
1  CIRCLE 50, 50, 48: CIRCLE 50, 50, 44:
   CIRCLE 200, 40, 37: CIRCLE 205, 38, 20
```

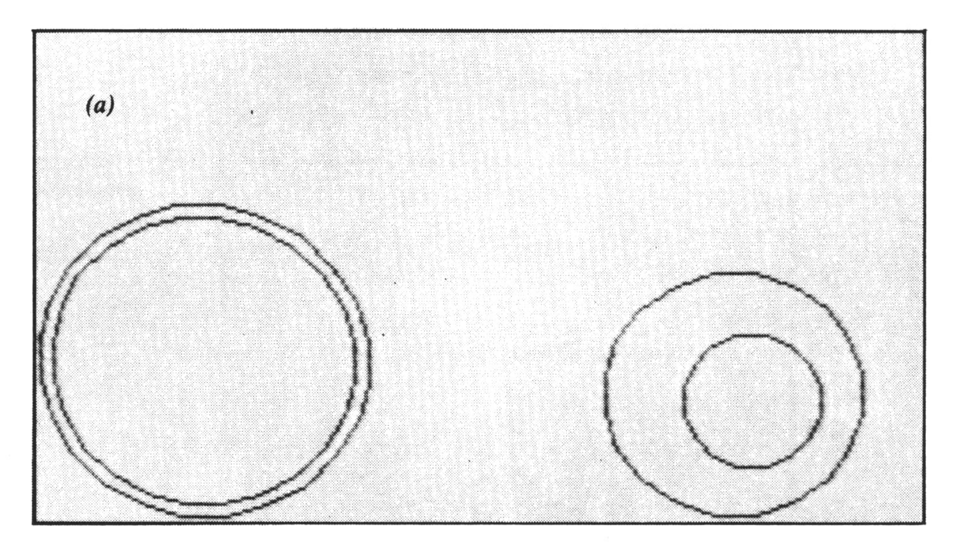
para tener algo que colorear. Ahora, pasa el programa. Es relativamente lento, pero parece que funciona, tal como demuestra la figura 2.6.

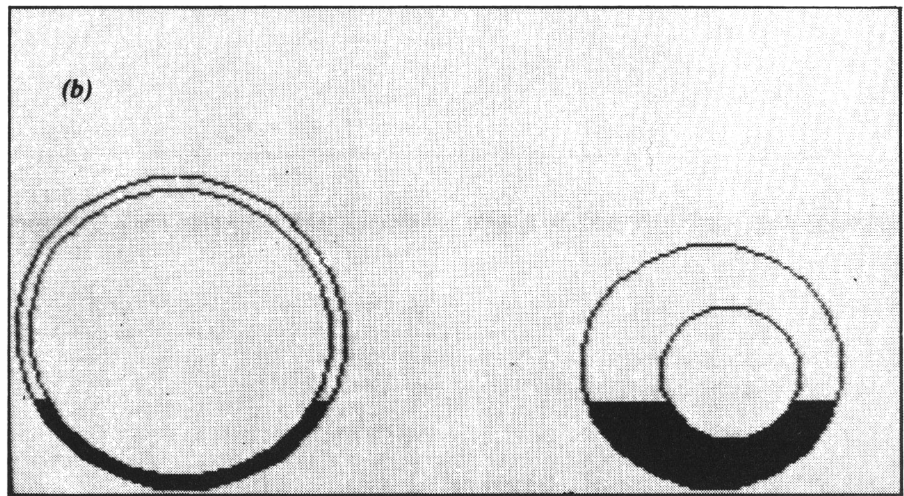
Ahora vienen las explicaciones. Las líneas 200-500 constituyen el programa principal: un bucle que comprueba a lo largo de cada línea horizontal (excepto la superior y la inferior) si hay o no trozos de curva. Si encuentra un trozo, va a la subrutina "rastrea" que sigue la curva a lo largo de la línea hasta encontrar los extremos



(como se marcaron en la figura 2.5); luego va a la rutina **prueba**, que decide, examinando la cuadrícula 3 x 3 alrededor de cada extremo, si la curva cruza o no la línea. Si sí lo hace, la rutina **lista** anota las coordenadas pertinentes.

Después de que se ha explorado una línea de esta forma, se rellena uniendo el primer punto al segundo, luego el tercero al cuarto, y así sucesivamente como sugerimos anteriormente. Sin embargo, hay una triquiñuela. Si rellenas la línea demasiado pronto, interfiere con la rutina **prueba** de la siguiente línea, y lo que obtienes es morralla. Así que la lista de los puntos se guarda primeramente en un **buzón** -la tabla a- y luego se transfiere a otra tabla b en la siguiente exploración, dispuesta para ser trazada. Las líneas 41ØØ-412Ø efectúan esta tarea.







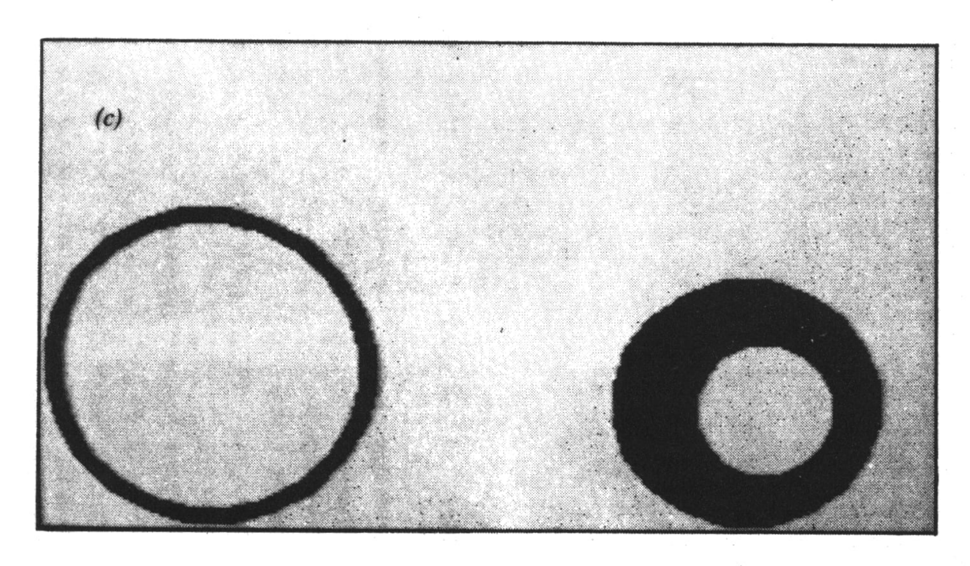


Figura 2.6 Coloreando la región entre los dos círculos de prueba: antes (a), durante (b), y después (c).

Para mantener sencillo el programa, se ha supuesto que las curvas a colorear **no tocan los bordes** (filas Ø y 175, columnas Ø y 255). Así que lo que se explora son las filas 1 a 174 desde la columna 1 hasta la 254.

EL MAPA

Hasta ahora todo va bien, pero ¿pasará la prueba decisiva? ¿Coloreará el mapa-mundi?

Carga tu mapa en memoria, usando

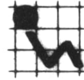
LOAD "mapita" SCREEN$

y luego tecleando

GO TO 1Ø

(**no** RUN, que arruinaría el mapa).

Ahora bien, tu mapa puede que no sea exactamente igual al mío. Lo que yo obtuve fueron las figuras 2.7 y 2.8.



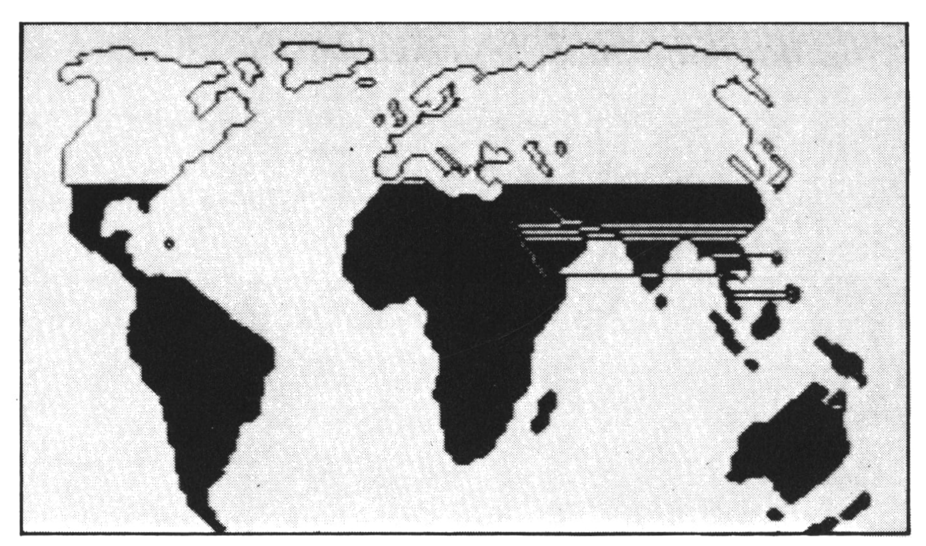


Figura 2.7 Coloreando un mapa de contorno del mundo -unas pocas pifias...

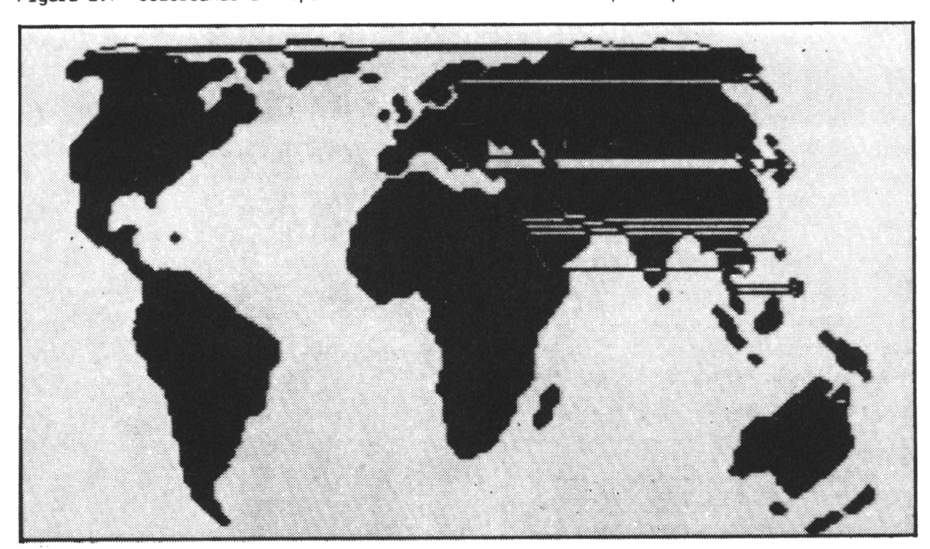


Figura 2.8 ...y unas cuantas más!

No está mal; pero no es perfecto tampoco. Hay fiebre malaria en Malasia y cierta confusión ruso-americana en las Aleutianas. Así que ¿qué es lo que ha fallado?

No es que haya pifias en la rutina de coloreado. Es más bien que la teoría sobre la que descansa, no funciona en ciertos casos... repugnantes. Examinando cuidadosamente el mapa, es posible observar que todos los sombreados defectuosos corresponden a uno de dos casos:

1. "extremos colgantes" como la figura 2.9a, o bien
2. "curvas tocantes" como la figura 2.9b.



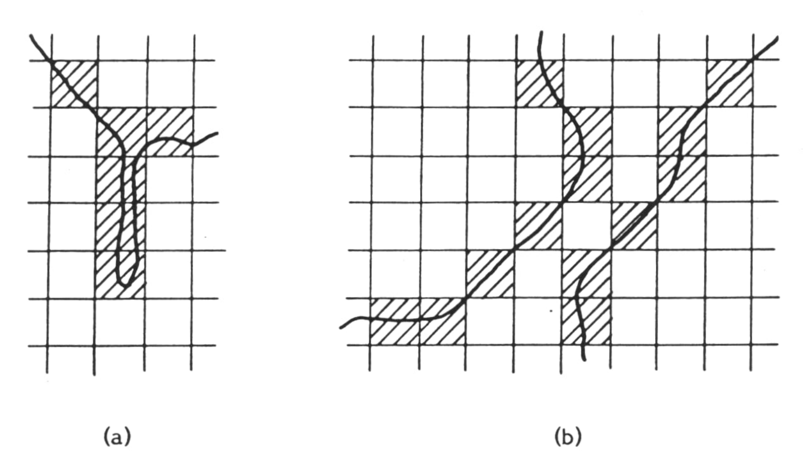


Figura 2.9 Las causas de las pifias: extremos colgantes (a), y curvas tocantes (b).

El método también se desmorona si las curvas se cruzan -simplemente, no está diseñado para manejar esa posibilidad.

Hay dos maneras de quitarlo. Una es refinar el programa para que busque extremos colgantes y curvas tocantes, y hacer algo con ellas: eso es difícil y requiere su tiempo. La otra es usar sólo curvas que no posean esos rasgos tan indeseables.

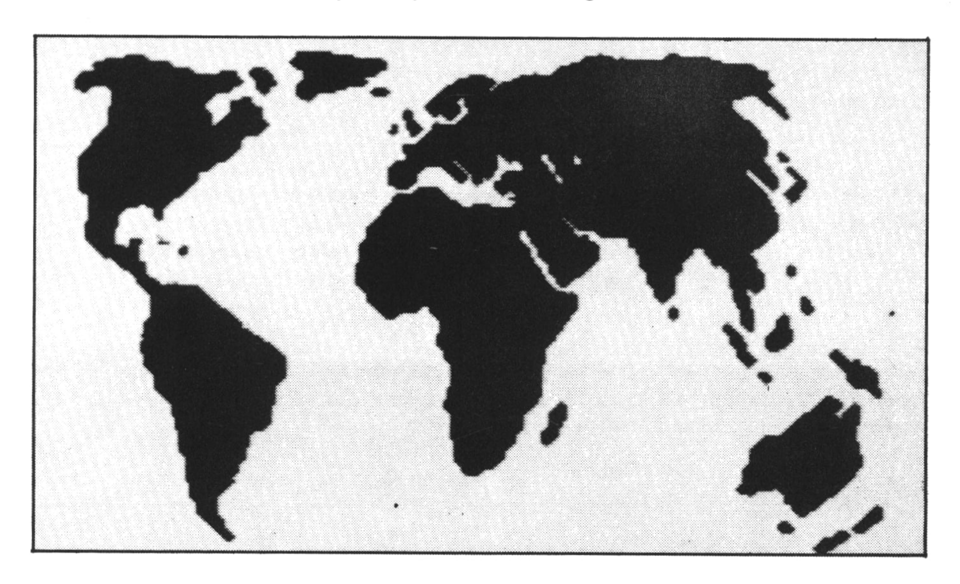
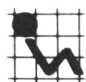


Figura 2.10 Depurado y completo.

Así que: si la rutina falla con tu mapa, como hizo con el mío, lo que debes hacer es usar el programa Dibujero para **modificar** el mapa, quitando los extremos colgantes y las curvas tocantes. Y **luego**, lo coloreas.



Posiblemente, todavía permanezcan unas pocas taras: yo no garantizo que no haya otras peculiaridades inoportunas. (Las interrupciones en la curva, causan estragos, pero realmente son también simples extremos colgantes). Pero serás capaz de percatarte cuando surgen, y usar el Dibujero para eliminarlas. Después de unas pocas pasadas, terminarás al final con algo como la figura 2.10. Cuando lo logres, guárdala bajo un nuevo nombre, digamos

SAVE "mapasólido" SCREEN$

y ya la tienes dispuesta para usar en otros programas.

Si crees que la forma es un poco rara, es porque el mapa del que partí fue una proyección equiárea Hammer, no la proyección Mercator, más habitual.

EL METODO DE INCENDIANDO LA PRADERA

Hay una aproximación totalmente diferente al programa de coloreado a la que puedes dedicar tus esfuerzos. La idea es dar al Spectrum una pista, "encendiendo un fuego" en una de las regiones del mapa con tierra. Para hacerlo, díle simplemente (por medio del teclado, o con una mota móvil) las coordenadas de ese punto. Por ejemplo, el punto 125,8Ø en el medio de Africa.

Ahora, deja que se desparrame el fuego: es decir, rellena todos los cuadritos adyacentes **a menos que tropieces con la línea de la costa.** Usando estos como nuevo punto de arranque, haz que progrese el fuego todavía más. Solamente para alcanzar la costa. De hecho, el fuego iniciado en Africa se desplegará a lo largo de Asia y de Europa al final. Nunca alcanzará el Reino Unido, ni América, ni Australia; así que tendrás que colocar "chispas" adecuadamente, para que también comience el fuego ahí... de hecho, necesitas un punto de ignición en cada masa de tierra conectada.

No es un método difícil de programar, pero el bosquejo anterior tiene que efectuarse de manera mucho más precisa. Si quieres un proyecto interesante para un programa, ahí tienes uno bueno.

COLOCANDO EL FUEGO EN EL MAR

Dado que el mar está bastante más conectado que la tierra, una forma mejor pudiera ser que fuera el mar por donde se desparramara el fuego (solamente tendrías que encenderlo separadamente para el Mar Caspio y el Mar Aral), habiendo estipulado que el color de la tierra corresponde al PAPER.



Si estás usando continuamente en el programa la misma expresión, con diferentes valores para las variables, no recurras a una subrutina -recurre a:

3 Funciones definidas por el usuario

Una función viene a ser como las "cajas negras". Tú le das algunas ristras numéricas o litéricas, -los argumentos-; ella te devuelve otras -los resultados-. Por ejemplo, si tú das como argumento de la función LEN la ristra "lítero" te devuelve el número 6 -la longitud de esa ristra- porque lo que ejecuta es

LEN "lítero" = 6

El Spectrum tiene un montón de funciones incorporadas como VAL, COS, TAN, EXP, y muchas otras. Pero algunas veces, te encuentras usando continuamente una expresión determinada, una y otra vez, sobre variables diferentes. Tú **puedes** realizar esto como una subrutina, lo que involucra habitualmente, montones de comandos LET a = 39: LET b = 21 ...antes de que puedas citar la subrutina.

La tecla DEF FN (modo E/tecla SYMBOL-tecla 1) te permite estipular tus propias funciones; y la tecla FN (modo E/tecla SYMBOL-tecla 2) úsalas. Cada una debe estar seguida de una sola letra: DEF FNa y FNa; o bienDEF FNb y FNb; y así recorriendo el alfabeto. Si el valor resultado es un literal, tienes entonces que usar FNa$, FNb$, etc. Puedes **usar** mayúsculas también; pero el Spectrum no te lo tiene en cuenta. Es decir, él considera idénticas FNa y FNA. Así que lo que tienes a tu disposición son 26 funciones de cada tipo, numérico y litérico.

Por ejemplo, supón que en nuestro programa permanentemente queremos sumar tres números y luego dividirlos por 3 para obtener la media. Tenemos en el programa montones de expresiones como (x + y + z) / 3 y (precio1 + precio2 + precio3) / 3 desparramadas por todo el listado. Por lo que estipulamos una función como ésta:

10 DEF FNa (p, q, r) = (p + q + r) / 3

Una vez que hemos hecho esto, podemos sustituir las expresiones mencionadas por:

FNa (x, y, z)

FNa (precio1, precio2, precio3)

siempre que aparecen en el programa.

Vamos a comprobarlo:

```
10 DEF FN a(p,q,r)=(p+q+r)/3
20 INPUT x,y,z
30 PRINT FN a(x,y,z)
```



Pasas este breve programa e impones valores a los argumentos como:

1	2	3	(resultado 2)
4	4	4	(resultado 4)
77	91	3	(resultado 57)

para asegurarte que funciona correctamente.

Son dignos de notarse varios puntos. Primero, las variables p, q, r que aparecen en la definición de función -llamadas argumentos o parámetros- son simplemente sitios de la memoria con ese nombre: puedes usar esas mismas letras en cualquier otra parte del programa sin que provoques ningún desastre, y puedes definir la **misma** función, usando otras letras, por ejemplo:

```
10 DEF FNa (a, b, c) = (a + b + c) / 3
```

No importa incluso, si la letra usada para denominar la función (en este caso "a") aparece también como argumento dentro de los paréntesis que definen la función. El Spectrum puede distinguir cuál es cuál.

Además, las letras de dentro de los paréntesis deben ser simples: no puedes usar variables como precio1 en la **definición.** Aunque no hay ningún percance si usas precio1 cuando la función se cita en otra parte del programa: FNa (precio1, precio2, precio3) es totalmente correcto.

Puedes tener variables litéricas en la definición, pero también deben ser únicamente una letra, seguida del signo $. Puedes mezclar números y líteros; y el ordenador es completamente feliz con expresiones como

```
10 DEF FNb (b, b$) = b * LEN b$
```

incluso aunque la "b" aparece en tres lugares con tres significados diferentes. Luego, al citar FNb se multiplica la longitud de la ristra b$ por el número b. Si tú citas luego

```
FNb (7, "sarta")
```

obtienes como resultado 35, que es lo que se encuentra al calcular

```
7 * LEN "sarta" = 7 * 5 = 35
```

La definición de una función **no** tiene por qué venir delante en el programa, antes de poderla usar; pero debe estar en alguna parte del programa (de la misma forma que las definiciones DATA).

Como otro ejemplo, ensaya la función con parámetros y resultados literales, siguiente:

```
10 DEF FNm$ (u$, v$) = u$ + u$ + v$
```

y determina lo que obtendrás cuando la apliques con los siguientes argumentos:

```
FNm$ ("B", "C")
FNm$ ("pa", "natas")
FNm$ ("ca", "huete")
FNm$ ("Bi", " □ Andersen")
```




Cada definición de función, debe incluir los paréntesis. Pero **puede no** tener argumentos dentro de ellos! Si escribes

```
10 DEF FNk ( ) = 77
```

siempre que cites FNk te devolverá el número 77. (hay otras ocasiones más sutiles donde realmente puede ser útil este tipo de cosas: aquí simplemente parece estúpido).

Además, la definición de la función puede incluir en la expresión, variables que no estén encerradas entre los paréntesis, por lo que no serán argumentos de la función; siempre que tengan un valor definido previamente en el programa. Por lo tanto:

```
10 DEF FNa (x) = x + q
20 LETq = 5
30 PRINT FNa (10)
```

da como resultado 15, pero

```
10 DEF FN a(x)=x+q
20 PRINT FN a(10)
30 LET q=5
```

da un mensaje de error. (AVISO: limpia las variables -pulsa CLEAR- antes de comprobar ésto; el valor de q todavía permanece en memoria como consecuencia del primer ensayo). Para comprobar que la definición realmente puede ir en cualquier parte, ensaya

```
10 LET q=5
20 PRINT FN a(10)
30 DEF FN a(x)=x+q
```

ALGUNOS USOS

Sólo merece la pena definir una función de esta forma si (1) continúas usando la misma expresión una y otra vez pero con diferentes valores de los argumentos o (2) tu programa manipula una función que no es siempre la misma -tal como un programa de gráficos que traza la curva de una función dada. En ese caso, puede ser preferible escribir por ejemplo, FNa para manipularla; y luego hacer que el usuario edite el valor deseado de la función (o trucos sigilosos con la función VAL, y puede usarse en instrucciones INPUT -con un coste: un programa más lento. Véase Capítulo 16).

Por ejemplo, la distancia entre los puntos (a, b) y (c, d) en la pantalla, tomando como unidad de distancia la mota gráfica, viene dada por:

```
10 DEF FNd (a, b, c) = SQR ((a - c) * (a - c) + (b - d) * (b - d))
```

que es la versión del Spectrum del Teoréma de Pitágoras. Si tienes un montón de distancias que calcular, esto puede ser útil. (No sólo en matemáticas: puedes tener un mapa de los Estados Unidos y desear conocer lo lejos que está Los Angeles de Oklahoma City).

Ten siempre en mente la posibilidad de usar **valores lógicos.** Recuerda que una aserción logica como "x < > 4" tiene, según el ordenador, un valor **numérico:** 1 si es cierta, 0 si es falsa. Por lo tanto

```
10 DEF FNo (u, v) = u > = 0 AND u < = 255 AND v > = 0 AND v < = 175
```



puede parecer a primera vista una tontería; pero para el Spectrum tiene claridad meridiana. De hecho, FNo comprueba si en una determinada posición (u, v) está dentro de los límites de la pantalla o no:

FNo (u, v) = 1 si y sólo si el punto (u, v) cae dentro de la pantalla

FNo (u, v) = Ø si y sólo si el punto (u, v) cae fuera de la pantalla

Así que, si necesitas comprobar a menudo ésto, es razonable considerar usar esta función.

O, consideremos las tarifas presentes para periódicos por correo aéreo fuera de Europa. Dependen del peso, y de la zona (A, B o C) en la forma siguiente:

	Zona A	Zona B	Zona C
Primeros 1Ø grs.	24	26	29
Cada 1Ø grs. adicionales o porción de ellos	11	14	15

Establezcamos una función definida por el usuario FNp que nos dé el precio para cualquier peso de w gramos y en cualquier zona z$ (= "a", "b", o "c"). Nos entregará un valor numérico, así que no tenemos que llamarla FNp$. Y será algo así como:

DEF FNp (w, z$) = algo bastante lioso...

Vamos a abordarlo gradualmente. Pensemos primero, simplemente en la zona A. Para mayor simplicidad, supongamos que el peso siempre es mayor de Ø. Por lo tanto, siempre se aplican los "primeros 1Ø gramos", así que ciertamente siempre pagaremos como mínimo 24 pesetas. El peso que queda, será w - 1Ø. Si esto es Ø o negativo, entonces ya está hecho; pero si no lo es, debemos redondearlo a los **siguientes** 1Ø gramos.

Para redondear un número n al siguiente múltiplo de 1Ø, podemos usar la expresión

-1Ø * INT (-n/1Ø)

Compruébalo: si n = 43, tendremos:

-n = -43

-n/1Ø = -4,3

INT (-n/1Ø) = -5 (sí, así es. Ensáyalo. La función INT redondea por abajo)

1Ø * INT (-n/1Ø) = -5Ø

-1Ø * INT (-n/1Ø) = 5Ø

que es lo que queríamos.

Como necesitaremos hacer esta tarea varias veces, vamos a **definir una función:**

DEF FNr (n) = -INT (-n/1Ø)

que es la función "redondeo" después de dividir por 1Ø. ¡Muy bien!



Ahora, en la zona A, el precio que pagaremos es

24 + 11 * FNr (w - 1Ø)

siempre que w > 1Ø, y sólo pagaremos 24 si w < = 1Ø. Hmmm... valores lógicos! Tendrémos que pagar

24 + (w > 1Ø) * 11 * FNr (w - 1Ø)

porque (w > 1Ø) adopta el valor 1 cuando w > 1Ø, y nos añade la porción de coste extra; pero adopta el valor Ø cuando w < = 1Ø, lo que nos deja simplemente las 24 pesetas.

Las zonas B y C tienen expresiones similares, pero con diferentes valores en lugar del 24 y del 11. ¿Cómo vamos a desarrollarlas?

Si usamos las letras minúsculas "a", "b", "c" para la variable zona z$, entonces los valores lógicos vienen de nuevo en nuestra ayuda. El número

24 * (z$ = "a") + 26 * (z$ = "b") + 29 * (z$ = "c")

adopta el valor 24 si z$ = "a", 26 si z$ = "b", y 29 si z$ = "c". (¿Por qué será?) Y de la misma manera podemos tratar la parte 11 - 14 - 15.

Todo eso nos lleva a las definiciones:

```
1Ø DEF FNr (n) = -INT (-n/1Ø)
2Ø DEF FNp (w, z$) = 24 * (z$ = "a") + 26 * (z$ = "b") + 29 * (z$ = "c")
    + (11 * (z$ = "a") + 14 * (z$ = "b") + 15 * (z$ = "c")) *
    (w > 1Ø) * FNr (w - 1Ø)
```

y así FNp (w, z$) nos da el precio del periódico, peso w, en la zona z$.

Proyectos

1. Establece FNt de manera que FNt (x) dé el coste de x latas de cerveza a 65 pesetas la unidad.
2. Define FNu de manera que FNu (x, p) dé el coste de x latas de cerveza a p pesetas la unidad.
3. Define la función FNj$ de forma que FNj$ (a$, b$, c$) entregue la ristra a$, o b$ o c$, que venga antes según el orden alfabético. (Nota: dos literales a$ y b$ están en orden alfabético si a$ es < = b$, según la notación del Spectrum para ordenamiento de valores literales).
4. Para los periódicos registrados en la dirección de correos, la tabla de precios anterior se convierte en:

	Zona A	Zona B	Zona C
Primeros 1Ø grs.	13	15	16
Cada 1Ø grs. adicionales o porción de ellos	3	4	5



Define FNq (w, z$) para que suministre el precio correspondiente a un periódico registrado, con peso w y zona z$.

5. Combina FNq y FNp (en el texto) para dar una función FNr (w, z$, y) siendo w el peso, z$ la zona, e y = Ø para periódicos no registrados, y = 1 para los registrados.

...hubo una vez un programador que siempre usaba tinta violeta porque le gustaba que sus programas permanecieran "inviolados"...

4 Caracteres de Control

Sin ninguna duda, habrás descubierto que la fila superior de teclas en tu Spectrum funcionan de forma diferente al resto, en cuanto a los modos y demás. Si no lo has hecho, ensaya el siguiente experimento. Pulsa sucesivamente:

NEW
CAPS SHIFT y SYMBOL SHIFT (para el modo ampliado: E)
La tecla 4, de la fila superior
"dedos"

Te encontrarás que tienes dedos verdes...

Al usar la fila superior de teclas en el **modo ampliado** (pulsando o no la tecla que elige mayúsculas) puedes, directamente a partir del teclado, fijar los colores, parpadear, ponerlo brillante, cambiar la posición en pantalla de lo expuesto, y así sucesivamente. El Manual da detalles completos de los efectos de determinadas combinaciones de teclas y modos; la parte importante para nosotros es:

Tecla	Efecto en modo extendido	
	Sin CAPS SHIFT	Con CAPS SHIFT
1	PAPEL azúl	TINTA azul
2	PAPEL rojo	TINTA roja
3	PAPEL magenta	TINTA magenta
4	PAPEL verde	TINTA verde
5	PAPEL cyano	TINTA cyano
6	PAPEL amarillo	TINTA amarilla
7	PAPEL blanco	TINTA blanca
8	BRILLO quitado	PARPADEO quitado
9	BRILLO puesto	PARPADEO puesto
Ø	PAPEL negro	TINTA negra



EFECTOS SOBRE LOS LISTADOS

Para propósitos de prueba, mete unas pocas líneas de programa:

```
10 REM
20 REM
30 REM
```

Ahora, sucesivamente pulsa:

(a) 1
(b) REM
(c) modo extendido/CAPS SHIFT y 9
(d) ENTER

Vigila cuidadosamente el cursor para asegurarte que realmente obtienes el modo extendido. Observarás que el programa **entero**, excepto la línea 1, parpadea. Lístalo; todavía sigue parpadeando.

Repite lo anterior, pero usa ahora diferentes teclas de la fila superior; y deja algunas veces fuera la tecla de elección de mayúsculas, en la acción (c). Hmmm... la pena es que afecta a **todo** el listado... ¿o no es así?

Vuelve a la versión parpadeante de la línea 1; y añade

```
11 REM [modo ampliado/4]
```

Ahora, todo lo que está después de la línea 11 se ha puesto verde (color 4). Pero todavía sigue parpadeando... ¿No había en la tabla anterior algo que quitara el parpadeo? Así que puede ser que necesitemos:

```
11 REM [modo ampliado/4] [modo ampliado/CAPS SHIFT y 8]
```

para sacarnos de encima el flash, pero mantener el verde desde la línea 20 en adelante. Ensaya y verás.

CARACTERES DE CONTROL

¿Qué es lo que está pasando?

Si consultas la lista de los códigos de los caracteres en el Manual, encontrarás un puñado de ellos al principio (número 6 a 23) que no pueden ser expuestos en pantalla (incluso ni como signos ?). Son los que llamamos caracteres de control y que afectan al funcionamiento del sistema.



He aquí la lista:

Código	Carácter
6	coma PRINT
7	EDIT
8	cursor a izquierdas
9	cursor a derechas
1Ø	cursor hacia abajo
11	cursor hacia arriba
12	DELETE
13	ENTER
14	número (usado en la organización del programa)
15	(no usado)
16	control INK
17	control PAPER
18	control FLASH
19	control BRIGHT
2Ø	control INVERSE
21	control OVER
22	control AT
23	control TAB

Estos caracteres, residen en la memoria como cualquier otro carácter, pero no aparecen expuestos en pantalla ni salen en los listados. Cuando usas la fila superior de teclas en el modo ampliado, lo que metes es alguno de estos caracteres. Por ejemplo, pulsando la tecla 4 con CAPS SHIFT tiene el efecto del carácter de control INK, con el color verde.

Aunque tú no **ves** en el listado un carácter de control, sí que ves su **efecto.** Y puedes comprobar que realmente está en la memoria, bien sea mirando lo contenido en la dirección pertinente (ver libro Programación Fácil), o bien mediante el siguiente experimento. Oprime sucesivamente

(a) 1

(b) REM

(c) modo E/1

(d) modo E/2

(e) modo E/3

(f) modo E/4

(g) modo E/5

(h) modo E/6

en que, con modo E nos referimos al modo ampliado o extendido. Observa que a diferencia del modo gráfico, tienes que elegir el modo E cada vez.

Verás que el cursor no se mueve, después de la REM. Simplemente cambia sucesivamente de color. Ahora usa DELETE, manteniéndolo pulsado para la repetición automática; observa cuánto tiempo tarda en efectuar su función a medida que pasa por todos esos caracteres de control.



Ensaya de nuevo, pulsando DELETE repetidamente en pulsaciones individuales; y vigila los cambios en la imagen. Obviamente, hay un montón de caracteres en la memoria que no aparecen en absoluto en la pantalla.

LISTADOS PARPADEANTES

Realmente, tú puedes **aprovechar** esta posibilidad; no es simplemente un truco bonito. Por ejemplo, puedes hacer que las sentencias REM se destaquen en un listado para mayor visibilidad:

```
 1  REM [modo E/CAPS SHIFT-9] Así se destaca [modo E/CAPS-8]
 1Ø REM
 2Ø REM
etc.
```

Lista este programa y comprueba que las sentencias REM continúan parpadeando. Ahora guárdalo en cinta, limpia la memoria, cárgalo de nuevo... y sí, todavía parpadea.

De forma similar, puedes destacar por colores las secciones de un programa, por ejemplo, las subrutinas. Impón los caracteres de control al comienzo de la primera línea de la rutina (después del número de línea -o también, al final de la línea anterior, ya que todo lo anterior a un número de línea queda ignorado). Todas las líneas subsiguientes aparecerán en el listado con ese color.

Para hacer un listado invisible, fija los colores de tinta y de papel al mismo valor. (Pero sigue listandose por impresora correctamente; o listará a partir de la línea siguiente a la que tiene el carácter de control, de manera que no te puedes proteger contra los piratas de esta forma. **Nada** sirve completamente de protección infalible ante los piratas; pero hay trucos, de los que éste es el más simple, para desanimar a los aficionados).

USO EN LOS PROGRAMAS

Puedes aprovechar en los programas los caracteres de control, para evitar tener que fijar la tinta, el papel, etc. por todas partes. Es útil especialmente, al crear imagenes llenas de color, titulares de páginas para los programas, y cosas similares.

Por ejemplo, para mostrar la bandera tricolor francesa en cualquier posición r (renglón) y c (columna), usa esto:

```
1Ø PAPER Ø: INK 7: BORDER Ø: CLS
2Ø INPUT r, c
3Ø FOR i = Ø TO 2
4Ø PRINT AT r + i, c; "(modo E/1) □□ (modo E/7)□□
   (modo E/2) □□ "
5Ø NEXT i
```

en que los cuadraditos, denotan el carácter ESPACIO o blanco.



Para obtener la bandera italiana, cambia el número 1 que figura en la línea 4Ø para que sea un 4.

No habrás dejado de notar que la ristra literal de la línea 4Ø se lista en su esplendor azúl-blanco-rojo.

Lo que nos lleva a algo un poquitín más ambicioso: la bandera de los Estados Unidos como una sola ristra de caracteres. Puedes sacar en pantalla una aproximación al pendón americano, usando caracteres de control, mediante la figura 4.1 y usando modo gráfico. Exige tiempo y cuidado; pero al final, comprenderás los caracteres de control tecleables directamente ¡enormemente bien!

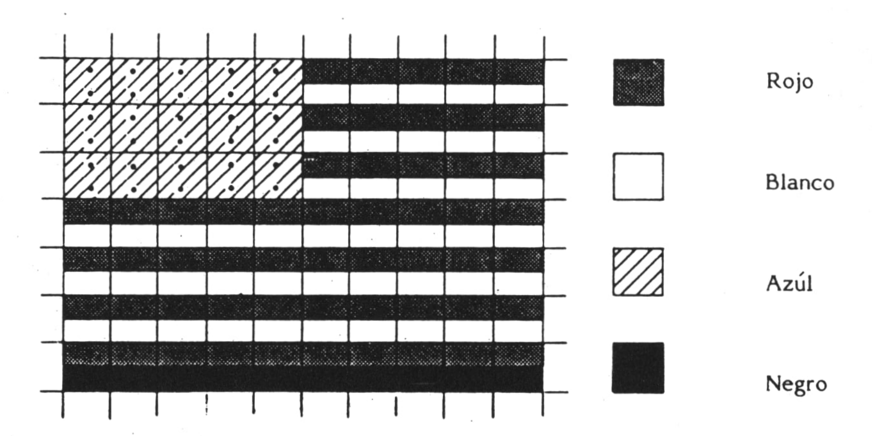


Figura 4.1 La bandera de EE.UU. impuesta directamente en el teclado como una larga ristra de caracteres.

Abajo te presentamos la cuenta golpe a golpe. Pronto verás cómo se va construyendo la bandera a medida que vas pulsando teclas, y serás capaz de anticipar lo que se necesita a continuación. Con un poco de práctica, puedes hacer este tipo de cosas partiendo de improviso, sin ningún bosquejo inicial. (g3c es en modo gráfico la tecla 3 con CAPS SHIFT).

```
1Ø  PRINT "(modo E/1) (modo E/CAPS/7) : : : : :
    (modo E/2) g3c g3c g3c g3c g3c (modo E/Ø)
    (22 espacios) (modo E/1) (modo E/CAPS/7) : : : : :
    (modo E/2) g3c g3c g3c g3c g3c (modo E/Ø)
    (22 espacios) (modo E/1) (modo E/CAPS/7) : : : : :
    (modo E/2) g3c g3c g3c g3c g3c (modo E/Ø)
    (22 espacios) (modo E/2)
    (g3c diez veces) (modo E/Ø)
    (22 espacios) (modo E/2) (g3c diez veces)
    (modo E/Ø) (22 espacios) (modo E/2) (g3c diez veces)
    (modo E/Ø) (22 espacios) (modo E/2) (modo E/CAPS/Ø)
    (g3c diez veces) (modo E/CAPS/7) (modo E/Ø)"
```




¡Vaya! Ahora, haz que el papel y el borde sea negro, la tinta blanca y pulsa RUN. La parte de las estrellas en la bandera puede ser mejorada (piensa un poco sobre gráficos definidos por el usuario) pero está claro lo que pretende ser. Y todo con una sola ristra de caracteres...

Para gráficos rápidos, llenos de color y en baja resolución, puedes usar esta técnica para convertir toda una pantalla llena de caracteres gráficos de colores en una sola ristra literal, que se expone casi instantáneamente. Así puede meterse cualquier dibujo que puedas plantear en una retícula de 64 x 44, directamente desde el teclado, como una ristra de 704 caracteres. Lleva su tiempo, y exige paciencia, pero es merecedor de una imagen atractiva y es un uso eficaz de la memoria (en la práctica, para facilitar el tecleo y la edición, te sugiero 5 ó 6 ristras de 128 ó 160 caracteres cada una).





"No es lo que quieres, es la manera en que lo haces". Los mismos datos, mostrados de diferente forma, pueden ser tan claros como el cristal o tan claros como la arcilla. Los gráficos y los colores añaden legibilidad. Pero ¿has pensado alguna vez usar una fórmula para tocar una canción?

5 Técnicas de imagen

El cómputo no es meramente un asunto de generar vastos tochos de estadillos numéricos, incluso aunque esto impresione a los visitantes burócratas. Es importante también, encontrar las formas adecuadas de presentar la información. En varias partes de este libro, yo exploro algunas de estas técnicas. Aquí, en este capítulo, considero cinco posibles formas de presentar series de números, producidas por un proceso matemático bastante interesante. El propio proceso se comenta al final, porque conozco un montón de gente que es menos aficionada que yo a las matemáticas...

El programa te pide que elijas el tipo de imagen que requieres, a partir de un menú de cinco opciones: luego debes teclear un número entre 0 y 1000. El listado es tan sencillo que te lo presento todo de una vez:

```
  10 DIM a(5)
  20 LET a(1)=40: LET a(2)=255:
     LET a(3)=704: LET a(4)=1000:
     LET a(5)=704
 200 PRINT 'Elige tipo de presentacion:
     1. Numerica
     2. Grafica
     3. Policroma
     4. Sonora
     5. Ambas'
 210 INPUT d
 300 PRINT ''Elige un numero entre 0 y□□□□1000'
 310 INPUT k: CLS
 320 LET k=k/250: LET x=.7
 350 FOR t=1 TO a(d)
 360 LET x=k*x*(1-x)
 370 GO SUB 1000*d
 380 NEXT t
 390 STOP
1000 REM numerica
1010 PRINT x,
1020 RETURN
```




```
2000 REM grafica
2010 IF t=1 THEN  PRINT k*250
2020 PLOT t,0: DRAW 0,170*x
2030 RETURN
3000 REM policroma
3010 PRINT  PAPER INT (8*x);'□';
3020 RETURN
4000 REM sonora
4010 BEEP .05,20-40*x
4020 RETURN
5000 REM ambas
5010 GO SUB 3000: GO SUB 4000
5020 RETURN
```

Antes de proseguir, puede que te guste ensayar esto. Cualquier número en la gama Ø-1ØØØ está permitido; pero los números mayores de 75Ø producen resultados más interesantes. La opción 1 expone una lista de números bastante poco significativa; la opción 2 produce algunas cosas llenas de picos bastante bonitas; la opción 3 dibuja barras y bloques coloreados por toda la pantalla; y la opción 4 interpreta tonadas bastante chocantes, algunas veces rítmicas y repetitivas, otras veces más complejas. La opción 5 combina las opciones 3 y 4.

ABORDANDOLO SISTEMATICAMENTE

Vamos a adoptar una aproximación más sistemática, eligiendo el valor del número a imponer, y comparando los tipos de imagen obtenidos. De hecho, adoptaremos cuatro valores standard,

766   88Ø   897   985

que entre ellos nos aclararán los puntos cruciales.

Pasa el programa con opción 1, y tecleando 766. Sacará una tabla de números en dos columnas. Leyendo sucesivamente los renglones, nos dan los valores sucesivos del número calculado por la línea 36Ø del programa. No es fácil de ver nada que tenga sentido (que es siempre el problema con las salidas numéricas tabuladas); pero el ojo entrenado notará que en cada columna los números se hacen más y más similares a medida que vas hacia abajo en la pantalla. La columna de la izquierda está más cerca de Ø.58, y la de la derecha de Ø.75. Así que los valores, alternativamente saltan de un valor (o aproximado) al otro. La secuencia de valores tiende hacia algo que es **periódico,** es decir, se repiten los mismos valores una y otra vez; y el **período** es 2.

Repite la pasada con opción 1, pero tecleando el siguiente número de prueba, 88Ø. Ahora el resultado es diferente: los números no acaban de asentarse en absoluto. Sin embargo, en cada columna están intentando alternar entre dos valores: Ø.82 y Ø.87 en la de la izquierda; Ø.51 y Ø.37 en la de la derecha. Toda la secuencia se repite cada **cuatro** pasadas, así que es una secuencia periódica con período 4.

Ahora ensaya la opción 1 con el 897. Hmmm, bien... ¿hay o no una cierta pauta? En la izquierda hay unos cuantos Ø.89; y varios cercanos al Ø.33 en la derecha; pero no está muy claro.

No te preocupes; la opción 1 con el 985 es todavía peor. De hecho, parece un total batiburrillo.



La opción 2 nos hace la vida más fácil. Para el número 766 nos da la figura 5.1, y el comportamiento de período 2 es muy claro a partir de la forma en que las espigas suben y bajan. Para el 88Ø, también es muy patente el período 4 (figura 5.2).

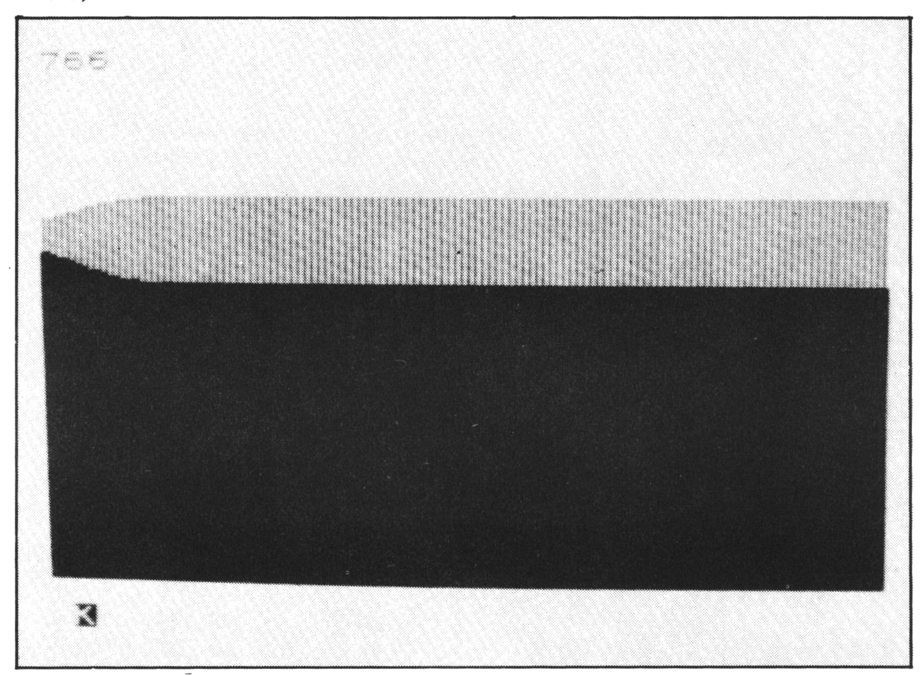


Figura 5.1 Imagen gráfica: k = 766. Período 2

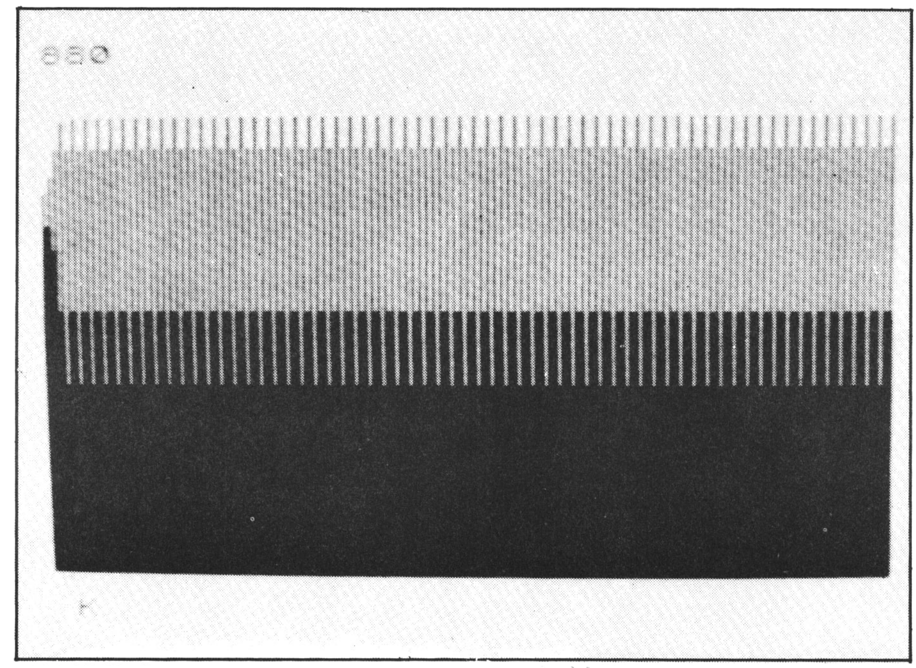


Figura 5.2 Imagen gráfica: k = 880. Período 4



Para el 897 hay rastros definidos de periodicidad, pero es un poco irregular (Figura 5.3).

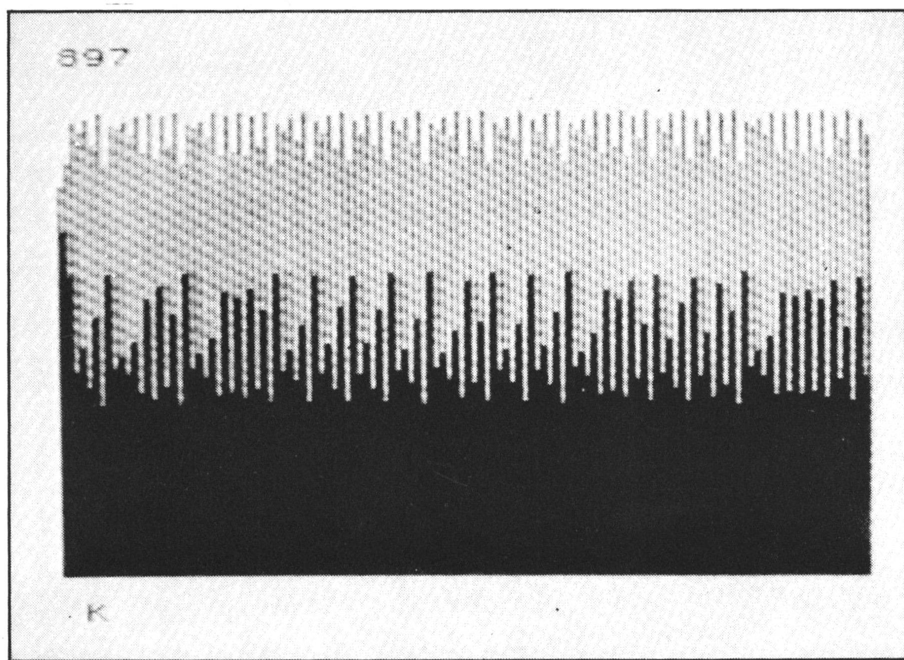


Figura 5.3 Imagen gráfica: k = 897. Permanecen rastros de periodicidad.

Para el 985 los picos son bastante más aleatorios (figura 5.4).

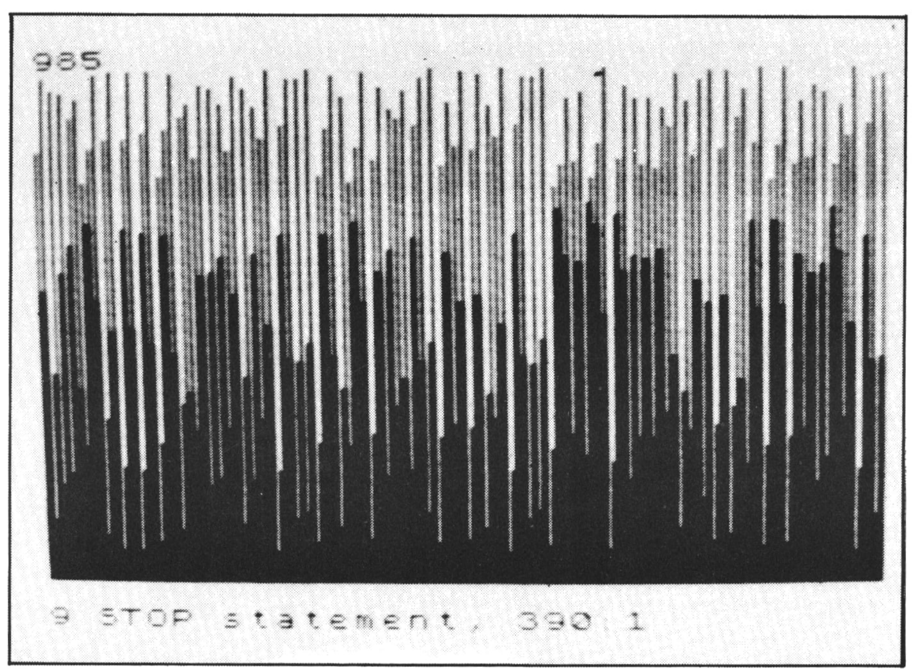


Figura 5.4 Imagen gráfica: k = 985. ¡El caos!



La opción polícroma (opción 3) resalta las periodicidades de manera mucho más impresionante -en parte porque los períodos 2, 4, etc. son divisores del número de caracteres en un renglón (32), lo que es una coincidencia afortunada.

Para el 766 verás barras verdes y cyanos (excepto muy al principio). Para el 88Ø (figura 5.5) las barras repiten cuádruplemente el patrón rojo-amarillo-verde-blanco. Para el 897 (figura 5.6) el efecto de las barras está definido claramente, con colores escogidos entre amarillo-magenta-blanco-verde-rojo; pero hay aquí y allá trozos en que un cuadrado saca un color diferente a la barra; y para el 985 (figura 5.7) lo que se obtiene es meramente cuadrados con apariencia aleatoria.

Figura 5.5 Polícroma (en el libro en blanco y negro). Las barras muestran periodicidad cuando k = 880.

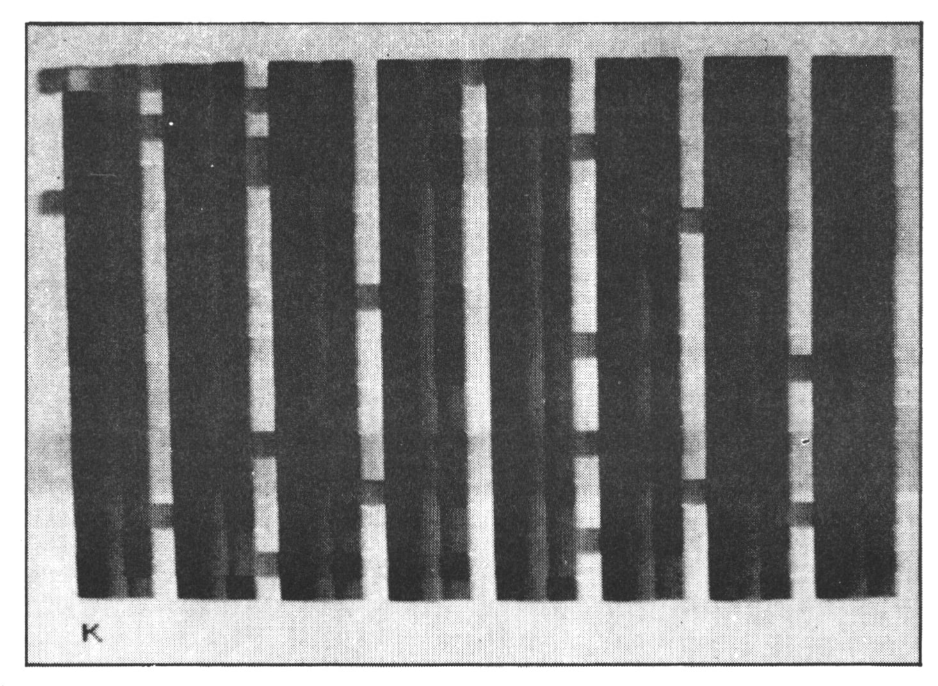


Figura 5.6 Polícroma (en el libro en blanco y negro): periodicidad parcial con interrupciones aquí y allá cuando k = 897.

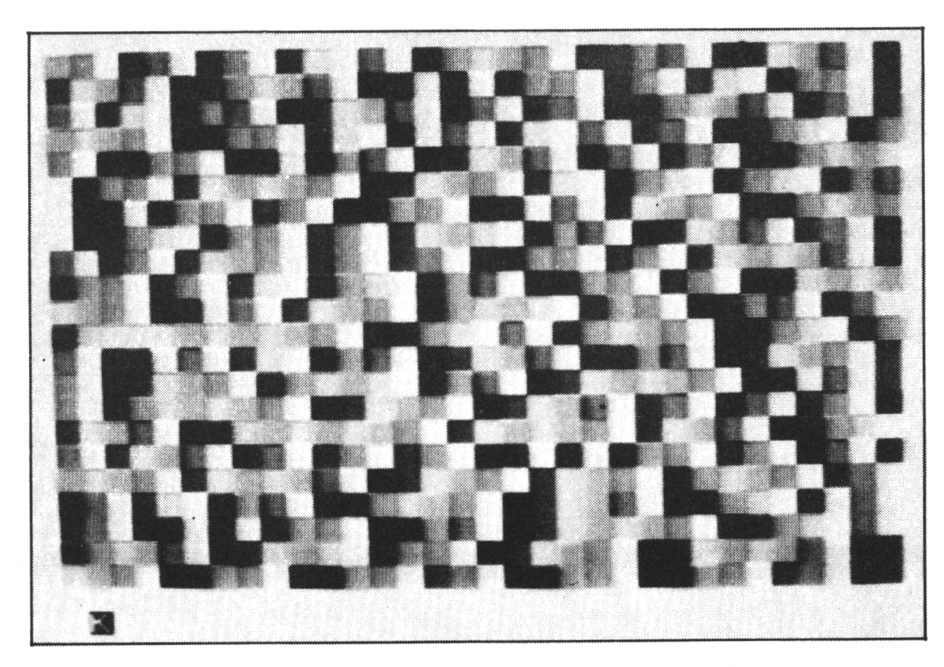


Figura 5.7 Polícroma (en blanco y negro en el libro): k = 985, el trazado aleatorio de los colores muestra el caos.



Uno normalmente, no piensa en presentar información mediante **tonadas**; pero aquí produce un resultado intrigante: el oído detecta la periodicidad como **ritmos.** Pasando el programa con la opción 4, el número 766, produce un sonido monótono du/da du/da (como una ambulancia pero con la sirena desafinada). Con el 88Ø el ritmo es mucho más compulsivo, repitiéndose una y otra vez un motivo de cuatro notas (lo puedes usar como acompañamiento a una canción pop). Con el 897 se obtiene una mixtura placentera de ritmo e irregularidad: ciertos motivos aparecen una y otra vez, pero a intervalos irregularmente espaciados. Con el 985, el sonido es como el producido por una orquesta al ensayar antes de la función.

La opción 5 nos permite ver los colores y escuchar el sonido conjuntamente; tienden a reforzarse uno al otro.

Ensaya otros valores para el número. Te encontrarás que con los números inferiores a 75Ø, todo conduce a un valor único (muy soso) y que el funcionamiento se hace más y más peculiar cuanto mayor es el número.

MODELOS REPRODUCTIVOS

Muy fascinante, sin ninguna duda. ¿Pero de qué trata todo esto?

El programa se basa en una fórmula usada en los modelos teóricos de desarrollo de poblaciones animales. Imagínate un lago lo suficientemente grande como para admitir como máximo 1ØØ hipopótamos. Dependiendo del ritmo reproductivo de cada generación, ¿cómo varía el número de hipopótamos? Con este programa particular, el número x que produce la salida numérica, gráfica o sonora, viene dado por

$$\frac{\text{número de hipopótamos en la presente generación}}{\text{número máximo posible } (= 1ØØ)}$$

y la tasa de reproducción es k/25Ø, siendo k el número que tú metes. Si esta tasa es menor que 1 (k < 25Ø) el número de hipos tiende a cero: la población de hipos se muere lentamente por falta de actividad reproductora.

Si está entre 1 y 3 (k entre 25Ø y 75Ø) el número tiende a un valor simple estacionario. Por encima de esas tasas, comienza a oscilar más y más abruptamente.

Por ejemplo, la secuencia con período 2 para k = 766, corresponde a tener 58 hipopótamos un año, 75 al siguiente, luego 58 de nuevo, luego 75 otra vez, y así sucesivamente. Lo que sucede es que la población de 58 es muy inferior a la que el lago puede sostener, así que el número de hipos aumenta; pero cuando lo hace se pasa del término medio ideal para dar 78, que es demasiado. Así que al año siguiente vuelve a caer -volviéndose a pasar por el lado contrario- hasta 58. Y el ciclo se repite.

El fenómeno aleatorio que se produce con k = 985 (y parcialmente con k = 897) es muy interesante matemáticamente, porque sabemos que no es **realmente** aleatorio en absoluto. Está producido por una fórmula muy simple: la línea 36Ø del programa. Pero ciertamente **parece** aleatorio. Se denomina **caos determinístico**, y es una espada de dos filos. Por un lado, muestra que sucesos aparentemente aleatorios pueden tener una estructura simple en el fondo; por el otro, fomenta las dudas sobre la capacidad de teorías aparentemente bien cimentadas hagan predicciones útiles. Este no es el lugar para hablarte sobre el caos. Pero si quieres saber más, te puedo recomendar el libro Conceptos de Matemática Moderna por Ian Stewart (no es que hagamos propaganda de la editorial Shiva, porque pertenece a la editorial Penguin Books).



La cuestión no es hurgar en la memoria para FISGAR el valor contenido en una determinada celdilla, o para HINCAR un determinado valor en una celdilla. La cuestión es dónde mirar (PEEK) y qué meter (POKE)!

6 Variables sistemales

La memoria del Spectrum, y no tengo ninguna duda de que ya lo sabes, viene en dos clases, **mal llamadas:** ROM (Real Only Memory) y RAM (Random Access Memory). Solamente en la RAM puedes cambiar el contenido de un lugar de la memoria (**dirección**). La mayoría de los sistemas operativos de la máquina, residen permanentemente en la ROM; pero hay una cierta cantidad que queda guardada en la memoria alterable (RAM) de manera que pueda cambiarse cuando sea necesario. Es el área de **variables sistemales,** y ocupa desde la dirección 23552 a la 23733. El Manual da una lista "comprensiva, pero no siempre comprensible" (comprensiva porque abarca todo aunque no sea fácil de comprender).

La mayoría de estas variables no son especialmente útiles, en cuanto a lo que se refiere a incorporarlas en los programas. Aunque tienen nombres dados en el Manual, solamente son para propósitos de referencia, pero no son nombres accesibles directamente desde BASIC.

Para "fisgar" el valor de una variable sistemal, usas PEEK. Para "hincar" el valor que tú quieres en ella, usas POKE (véase **Programación Fácil).** Para más detalles, continúa leyendo.

La mira de este capítulo es describir aquellas variables del sistema que probablemente encuentres provechosas y de darte unas cuantas ideas de cómo usarlas. La cosa importante radica en que las variables del sistema están **ahí,** y eres libre de acomodarlas a tu voluntad cuando surge la ocasión.

RESPUESTA DEL TECLADO

Tabla 6.1

Nemónimo	Dirección	Valor standard
REPDEL	23561	35
REPPER	23562	5
PIP	23609	Ø



Estas controlan el tiempo que el ordenador espera para efectuar la auto-repetición de una pulsación; la velocidad con que lo repite; y la longitud del pitido que se produce al pulsar una tecla. Para un teclado que responda más rápido con pitidos audible, mete (en modo comando o desde el programa):

```
POKE 23561, 1Ø
POKE 23562, 1
POKE 23609, 5Ø
```

Puedes variar los números para que sean justo a tu gusto: las gamas sensibles parecen ser 1Ø-2Ø; 1-3; 4Ø-1ØØ.

Para evitar tener que teclearlas cada vez que enciendes, puedes guardarlas en una cinta, y cargarlas nada más enchufar. Desde luego, se tarda más tiempo que tecleándolas -pero si también incluyes en la cinta tus utensilios favoritos (renumeración de líneas, véase capítulo 12; cambio de atributos, véase capítulo 7; algunos gráficos habituales definidos por el usuario) puede constituir un módulo muy útil para tenerlo asentado en el área BASIC.

ORGANIZACION DE LA MEMORIA

La memoria del ordenador está dividida en parcelas con secciones de programa que efectúan diferentes labores. Los linderos o límites entre estas parcelas pueden variar a lo largo de la sesión: las direcciones de estas cotas o límites se conservan como variables del sistema y te las doy en la tabla 6.2.

Tabla 6.2

Nemónimo	Dirección	Valor después de NEW (máquina 16K)
(16384: comienzo de fichero de grafismos)		16384
(22528: comienzo de fichero de atributos)		22528
(23296: comienzo de silo de impresora)		23296
(23552: área de variables sistemales)		23552
(23734: utilizado para ductora de discos)		23734
CHANS	23631-2	
PROG	23635-6	
VARS	23627-8	
E-LINE	23641-2	
WORKSP	23649-5Ø	
STKBOT	23651-2	
STKEND	23653-4	
RAMTOP	2373Ø-1	32599
UDG	23675-6	326ØØ
P-RAMT	23732-3	32767



Todas estas son variables de 2 octetos. Esto significa que por ejemplo, para encontrar dónde comienza el programa, debes usar

PRINT PEEK 23635 + 256 * PEEK 23636

ya que la dirección de comienzo del programa está en la variable PROG cuya sede es 23635 y 23636. En general, será (primer octeto) + 256 * (segundo octeto)

A la inversa, para cambiar PROG a digamos 13244 (y no te recomiendo ésto, es meramente para dar una idea general) tienes que desarrollar 13244 en la forma (octeto) + 256 * (octeto). De hecho, el segundo octeto (el **senior**) viene dado por

INT (13244/256), que arroja el valor 51

mientras que el primer octeto (el **junior**) mediante

13244 - 256 * INT (13244/256), que nos da 188.

Así que el comando sería

POKE 23635, 188: POKE 23636, 51

Igualmente, para cambiar RAMTOP al valor n (veremos más adelante las razones para querer hacerlo) usarás

POKE 23730, n - 256 * INT (n/256): POKE 23731, INT (n/256)

Solamente pueden aprovecharse RAMTOP y UDG para **hincarles** un valor; pero todo el conjunto de variables sistemales puede ser **fisgado** para determinar el sitio de la memoria en que cada sección comienza.

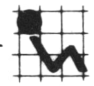
CHANS indica el comienzo del sistema de comunicaciones con la ductora de microdiscos, y no la encontrarás de mucha utilización. PROG es el comienzo del área donde está alojado el programa en BASIC: necesitas saber su valor para (e.g.) la rutina de renumeración de líneas del capítulo 12. VARS señala el lugar donde se guardan las variables: si quieres una rutina de renumeración de líneas perfecta, puede que encuentres uso de esta variable. O, si eres bastante persistente, puedes escribir una rutina que suprima de la memoria variables específicas (lo que sería como una instrucción de limpiar memoria pero local); aunque en estos casos sería mejor el código máquina. Desde E-LINE hasta STKEND son variables de más interés para el Spectrum que para su usuario.

Sin embargo, es importante el recinto de memoria entre STKEND y RAMTOP: ya que es la cantidad de memoria libre y disponible. (Realmente, la máquina coloca ahí dos "perchas" para colgar direcciones (véase **Código Máquina** y **Mejor BASIC**). la una para uso por el procesador Z80, la otra para las direcciones de vuelta cuando el programa va a ejecutar una subrutina; pero ambas son habitualmente bastante pequeñas). Así, como estimación, con precisión de unas pocas docenas de octetos, la memoria que está de repuesto, nos viene dada por

PRINT PEEK 23730 + 256 * PEEK 23731 - PEEK 23653 - 256 * PEEK 23654

y puedes incorporar este comando dentro de un programa. De hecho, 256 * (PEEK 23731 - PEEK 23654) es más simple y lo suficientemente aproximada.

RAMTOP tiene normalmente su sede al comienzo de los gráficos definidos por el usuario; pero se la puede bajar para conseguir espacio y meter aquellas cosas que no quieres que el sistema BASIC te machaque. Por ejemplo, rutinas en Código Máquina (véase Código Máquina y Mejor BASIC) o caracteres extra definidos por el usuario (los veremos más adelante).



Lo que se mete por encima de RAMTOP no se ve afectado al borrar el programa mediante NEW. (Tampoco se transfiere a la cinta al guardar el programa; pero puedes usar la opción de guardar octetos para conseguirlo, tal como explica el Manual).

UDG señala dónde comienza el área de gráficos definidos por el usuario. La razón principal para querer hincar otro valor aquí, es que andes corto de memoria y no quieres todos los 23 caracteres definibles. En ese caso, incrementas este valor para liberar el espacio extra que buscas.

RELOJ INCORPORADO

La variable del sistema FRAMES cuenta el número de "cuadros" de televisión que se han presentado desde la última vez que se encendió el ordenador. Recuerda que el barrido en la televisión es de 5Ø cuadros por segundo, de manera que efectivamente, tenemos incorporado un reloj. El Manual, trata este tema bastante extensamente, así que no vamos a repetir aquí la descripción; pero el punto primordial a observar es que esta variable tiene **tres** octetos, por lo que su valor numérico viene dado por

PEEK 23672 + 256 * PEEK 23673 + 65536 * PEEK 23674

y el número de segundos transcurridos es lo que salga arriba, dividido por 5Ø. Nota que la dirección 23674 sólo cambia cada 65536/5Ø = 131Ø.72 segundos, o sea, cada veinte minutos. Así que para la mayoría de las aplicaciones (como en la siguiente) puedes prescindir del tercer octeto y usar únicamente los dos primeros.

Y ahora, un programa para comprobar tu tiempo de reacción:

```
 10 PRINT "Prueba de reflejos"
 20 RANDOMIZE
 30 PAUSE (100+300*RND)
 40 LET t0=PEEK 23672+256*PEEK 23673
 50 PRINT AT 10,15,"!YA!"
 60 IF INKEY$="" THEN  GO TO 60
 70 LET t1=PEEK 23672+256*PEEK 23673
 80 LET t=t1-t0
 90 IF t<0 THEN  LET t=t+65536
100 PRINT AT 15,2;"Tu tiempo de reaccion es de□";t/50;"□segundos"
```

La línea 9Ø se preocupa del caso (improbable pero posible) en que el tercer octeto de FRAMES se haya incrementado en 1 durante el período de reacción.

Para usar el programa, mételo y pásalo, y en cuanto aparezca "¡YA!" en la pantalla, pulsa cualquier tecla. (Si eres fullero y mantienes una tecla continuamente pulsada, puedes luego aprovechar para modificar el programa e impedir eso).



EL REPERTORIO DE CARACTERES

La variable sistemal CHARS contiene la dirección donde comienza el **repertorio de caracteres:** una tabla de Øs y 1s que define la forma de los símbolos visibles. Diciéndolo más exactamente, contiene esa dirección citada menos 256: y la tabla comienza con el símbolo correspondiente al código 32, que es BLANCO o ESPACIO.

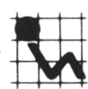
Normalmente, CHARS adopta el valor 1536Ø. Las instrucciones para exponer el símbolo con código n están contenidas en los octetos 1536Ø + 8 * n a 1536Ø + 8 * n + 7, y da las ocho filas del cuadrado 8 x 8 que ocupa ese símbolo. La forma es en binario, de la misma manera que para los caracteres definidos por el usuario (Fácil Programación página 49).

Puedes fisgar en estas direcciones para ver cómo la forma de un símbolo determinado está grabada en la ROM, y cómo se construye en la pantalla; usando este programa:

```
 10 INPUT 'Grafismo?';c$
 20 IF c$='' OR LEN c$>1 OR CODE c$<32 THEN  GO TO 10
 30 LET n=CODE c$
 40 FOR i=0 TO 7
 50 LET x=PEEK (15360+8*n+i)
 60 LET p$=''
 70 FOR t=1 TO 8
 80 LET xs=INT (x/2): LET xj=x-2*xs
 90 LET p$=('*' AND xj)+('.' AND NOT xj)+p$
100 LET x=xs
110 NEXT t
120 PRINT p$
130 NEXT i
```

Este programa, recupera los números binarios de la ROM y convierte el Ø en un punto y el 1 en un asterisco. Así que la letra "a" da:

Dirección en ROM	Contenido en binario	Efecto gráfico
16136	Ø Ø Ø Ø Ø Ø Ø Ø	. . . .
16137	Ø Ø Ø Ø Ø Ø Ø Ø	. . . .
16138	Ø Ø 1 1 1 Ø Ø Ø	. . * * * . . .
16139	Ø Ø Ø Ø Ø 1 Ø Ø	. . . . . * . .
1614Ø	Ø Ø 1 1 1 1 Ø Ø	. . * * * * . .
16141	Ø 1 Ø Ø Ø 1 Ø Ø	. * . . . * . .
16142	Ø Ø 1 1 1 1 Ø Ø	. . * * * * . .
16143	Ø Ø Ø Ø Ø Ø Ø Ø	. . . .



¿Distingues el tipo de "a" entre las estrellitas?

Los puntos y estrellitas son por claridad. Para obtener el efecto verdadero, cambia "." a "□" y "*" a "■".

Puedes usar esta técnica en los programas para producir caracteres visivos ocho veces su tamaño habitual. Sustituyendo cada cuadrito por un cuadrito 2 x 2, puedes llegar a 16 veces el tamaño original; un poquito apelmazado, pero dramático. Manipulando los ocho números binarios en bloques de 2 x 2, puedes traer a escena los caracteres gráficos y producir letras 4 veces el tamaño habitual (véase en el reverso) e inventando gráficos adecuados definidos por el usuario, puedes producir símbolos al doble del tamaño habitual.

Más chocante es el efecto de alterar el valor de la variable CHARS.

Mete un programa que tenga aproximadamente una página de listado para un efecto mejor. Teclea directamente el comando

```
POKE 23606, 2
```

Hmmm... es un poquito raro: los caracteres se han desplazado un par de líneas y parece que se han roto un poquito. Ahora ensaya

```
POKE 23606, 8
```

(e ignora los mensajes que aparecen en la pantalla, que también son extraños). Ahora, has vuelto a obtener letras bonitas, pero el listado parece que está cifrado de alguna manera... De hecho, cada letra ha sido cambiada por la siguiente en la tabla de caracteres; porque hemos atontado al ordenador y está mirando 8 octetos por delante de donde debiera estar haciéndolo.

Para **verdadera** diversión, teclea

```
POKE 23607, 50
```

Un uso juicioso de este comando en un programa, haría que los listados fueran incomprensibles. Sin embargo, un poco de trabajo detectivesco por parte del propenso a pirata volvería a editarlo de nuevo... La vida es triste...

Para volver a sacar una imagen comprensible, teclea

```
POKE 23606, 0: POKE 23607, 60
```

pero lleva cuidado, ya que no podrás ver reflejado en pantalla lo que estás tecleando hasta que todo termine. Quitando el enchufe un momento, restaurará las condiciones iniciales si todo lo demás falla.

Todo esto es muy fascinante, ¿pero adónde vamos? El punto primordial es que puedes dotar al ordenador de repertorios de caracteres completamente nuevos -tantos como sitios tengas en la memoria- en lugar de solamente los 23 caracteres definidos por el usuario. La forma de hacerlo se explica en el capítulo 15, porque haría demasiado largo este capítulo en este momento; la idea es construir el nuevo repertorio de caracteres en otra parte de la memoria que el sistema BASIC no use, bajando el valor de RAMTOP y fijando los códigos para los rasgos de cada carácter. A continuación cambiamos el valor de la variable CHARS y todos nuestros nuevos caracteres se hacen accesibles. Si volvemos a hincar el valor antiguo, el viejo repertorio aparecerá. Así que con dos diminutas subrutinas que hinquen uno u otro valor, más 2048 octetos de datos, y ya posees 256 nuevos caracteres -símbolos gráficos, matemáticos, o lo que sea. Desde luego que no tienes que pasar por todo el trago: se pueden estipular menos de 256 caracteres usando el mismo método, si lo prefieres.



Usando los caracteres de dibujar (la fila superior de teclas) puedes dibujar símbolos a cuatro veces su tamaño normal, que es suficiente para llamar la atención. Este es un programa que acepta cualquier ristra de grafismos con longitud 8 o menor, y la amplía cuatro veces. (El límite en la longitud es simplemente para que el resultado se adecúe a la pantalla: puedes modificarlo fácilmente si quieres escribir varios renglones. Está basado en la manera inteligente en que la gente de Sinclair (por esta vez!) ha codificado los símbolos para dibujar gráficos. Para sacar el código correspondiente a un símbolo, suma adecuadamente los números de esta cuadrícula, que

2	1
8	4

corresponden a cuadritos negros, y luego le añades 128. Por ejemplo ▚ tiene como código 2 + 4 + 128 = 134. Puedes comprobarlo en las páginas del Manual.

```
 10 DIM q(8,8)
 20 LET i0=PEEK 23606+256*PEEK 23607
 30 INPUT c$
 40 LET s=LEN c$: IF s>8 THEN  LET s=8
 50 FOR r=1 TO s
 60 LET i=i0+8*CODE c$(r)
100 FOR a=0 TO 7
110 LET m=PEEK (i+a)
120 FOR b=0 TO 7
130 LET q(a+1,8-b)=m-2*INT (m/2)
140 LET m=INT (m/2)
150 NEXT b
160 NEXT a
200 FOR u=1 TO 4
210 LET t$=""
220 FOR v=1 TO 4
230 LET k=q(2*u-1,2*v)+2*q(2*u-1,2*v-1)+4*q(2*u,2*v)+8*q(2*u,2*v-1)
240 LET t$=t$+CHR$ (128+k)
250 NEXT v
260 PRINT AT u,4*r-4;t$
270 NEXT u
280 NEXT r
```



La figura 6.1 muestra el resultado de teclear "Testing?".

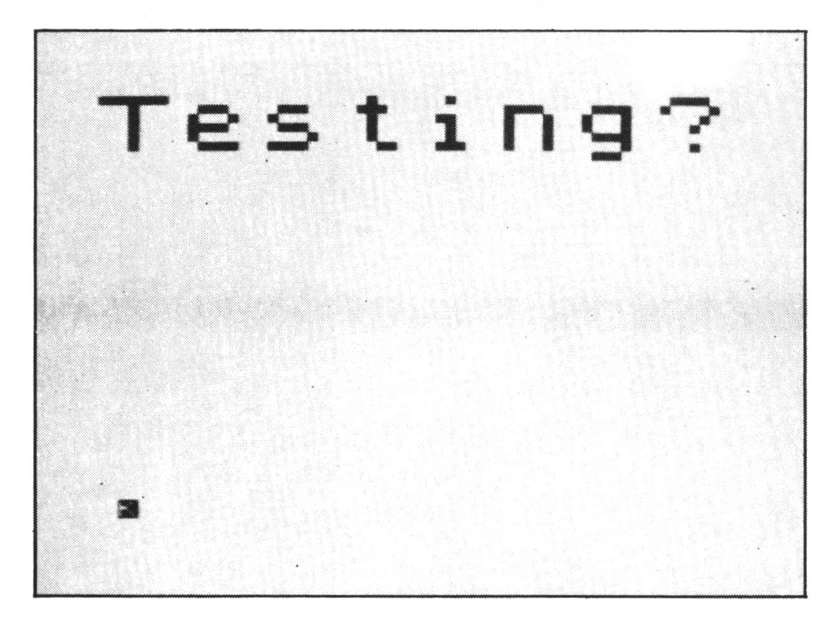


Figura 6.1 Probando los caracteres a tamaño cuádruple.

SALTOS IMPOSIBLES

El Spectrum te permite ahorrar montones de números de línea (una de las mayores objeciones al BASIC es por qué usa números de línea) escribiendo varias instrucciones en una sola línea:

```
10 LET a=1:LET b=49:PRINT a+b:GO TO 10
```

(por ejemplo). Está muy bien y es bueno, pero solamente puede saltar mediante la sentencia GO TO o GO SUB a la **primera** instrucción de tal línea.

A menos que alteres las variables sistemales NEWPPC y NSPCC, en las direcciones 23618-9 (2 octetos para NEWPPC) y en 23620 (1 octeto para NSPCC). Así se fuerza un salto a la línea señalada en NEWPPC, y al número de instrucción dentro de la línea, señalado en NSPPC.

Mejor que elaborar la teoría, aquí hay un programa que aclara todo.

```
 3 INPUT a
 5 POKE 23618,10: POKE 23620,a
 7 PRINT 'Se supone que no vendria aqui'
10 PRINT '1': PRINT '2': PRINT '3'
```

Ejecuta este programa, e impón 1 como valor de a; ensaya otra vez con a = 2 y a = 3.

A la línea 7 no se llega en ningún momento. El primer comando de la línea 5, fuerza un salto a la línea 10. Si a vale 1, el segundo comando de la línea 5 hace que el salto sea a la primera instrucción de la línea 10; si es un 2 hace que sea a la segunda instrucción; y si es 3 a la tercera.



En general, se fuerza un salto mediante el comando

POKE 23618, nj: POKE 23619, ns: POKE 23620, a

que tiene el mismo efecto que la instrucción

GO TO nj + 256 * ns, a-ésima instrucción

(si es que este comando fuera posible en BASIC, que no lo es).

Una incomodidad: las instrucciones condicionales IF/THEN. La parte THEN es tratada por el ordenador como si fuera una instrucción separada en la línea, que ha de incluirse además en la cuenta de instrucciones de esa línea. Y si saltas a la parte THEN, usando NSPPC, obtienes un mensaje de error indicándote "Sin sentido en BASIC". Para ver lo que sucede, ensaya con esto:

```
 2 INPUT x
 3 INPUT a
 5 POKE 23618,10: POKE 23620,a
 7 PRINT "Se supone que no vendria aqui"
10 IF x=0 THEN  PRINT "1": PRINT "2": PRINT "3"
```

El ordenador considera que la línea 10 tiene **cuatro** comandos:

(IF x = 0) (THEN PRINT "1") (PRINT "2") (PRINT "3")

Así que ensaya con 1, 2, 3, 4 como valores de a. Poniendo x = 0 hace que la condición sea cierta; x = 1 que sea falsa. El resultado es algo como esto:

x	a	Expuesto en pantalla
0	1	1 2 3 (en columnas)
	2	Sin sentido en Basic
	3	2 3
	4	3
1	1	(blanco -condición no cierta)
	2	Sin sentido en Basic
	3	2 3
	4	3

Observa que incluso con x = 1, los saltos al tercero o cuarto comando de la línea, produce la exposición en pantalla: la cláusula IF/THEN queda ignorada (como si las instrucciones estuvieran en líneas separadas, y hubieras usado el salto adecuado).

EL FONDO DE LA PANTALLA

Los atributos para la parte inferior de la pantalla (donde salen los mensajes de error) son fijados por la máquina; y en ocasiones, puedes terminar con un bloque de color en el fondo que no concuerda con el del borde, si has cambiado el del borde durante el paso del programa. La variable sistemal BORDCR con sede en 23624, contiene el valor 8 * c, siendo c el color.



Punzando en esa dirección otro valor, no produce un efecto inmediato; pero si no te importa poner a Ø tus variables, puedes restaurar el color mediante

POKE 23624, 8 * c: CLEAR

Puede que encuentres uso para esto. Mira también el capítulo 7 sobre ficheros de grafismos y atributos.

Siguiendo todavía con el asunto de la sección de mensajes en la pantalla, hay otra manera de accesarlo usando un comando diseñado para la microductora de disco. Ensaya este programa (el signo "#" -diésis*- está en la tecla 3, simultáneamente con la tecla de cambio de símbolos).

```
10 PRINT #0;'Hola tronco'
20 GO TO 20
```

(La última de estas instrucciones la he puesto simplemente para impedir que aparezca el mensaje "OK" y arrase con todo). Usando PRINT # Ø puedes exponer algo dentro de la sección de mensajes. Hay algunos caprichos: se tiende a perder la línea 21 de la sección principal, o que se **desrrolle** la imagen cuando no lo esperas. El mensaje sacado con PRINT # Ø puede tener más de 64 caracteres: se amplía la sección inferior y se despliega hacia arriba. Usando comandos de color, o caracteres de control, puedes conseguir además color con esta acción.

Si tienes acoplada una impresora, intenta usar en lugar de esto el comando PRINT # 3. Esta vez, ves que el mensaje se envía a la impresora -lo mismo que si hubiera sido LPRINT.

PRINT # 1 también produce mensajes en la parte inferior de la pantalla, y PRINT # 2 los produce en el sitio habitual -como un PRINT normal-. Los otros comandos PRINT # n, con n > 3, son los que conciernen al sistema de minidiscos.

Es provechoso cuando quieres exponer una serie de mensajes (por ejemplo) saber usar las opciones de pantalla o de impresora. Usa

PRINT # n; "Un mensaje cualquiera"

Y luego haciendo n = 2 lo sacas por pantalla, y con n = 3 por impresora. Lo cual es mucho mejor que la alternativa

IF n = 2 THEN PRINT "Un mensaje cualquiera"

IF n = 3 THEN LPRINT "Un mensaje cualquiera"

DESROLLANDO

Por lo que a mí respecta, el rasgo más incómodo del Spectrum es la forma en que pregunta si desrrolla o no. Tienes que pulsar constantemente el teclado para hacer que desrolle, o que deje de desrollar; y el mío termina habitualmente desrollando cuando no quiero que lo haga y parando cuando quiero que desrolle. El sistema es claramente perverso, y me hubiese gustado que Sinclair hubiera trabajado un poquito más sobre él.

Si deseas forzar desde un programa el desrolle automático, puedes usar la variable sistemal SCR-CT, con sede en 23692. Contiene el número de líneas que quedan antes de que se muestre la solicitud de desrolle.

* "diésis" es delimitador o separador en general. Al "#" se le llama vulgarmente "almohadilla".



Por lo tanto, el comando

POKE 23692, 2: PRINT AT 21, 31: PRINT

provoca que se desplace una línea hacia arriba, **sin** solicitar confirmación por el teclado. (Ya mencioné esto en **Programación Fácil,** pero merece mencionarse de nuevo para que todo sea completo).

COORDENADAS DE PUNTEO

También las he mencionado: aplica el mismo razonamiento anterior. La variable sistemal COORDS contiene en sus 2 octetos, las coordenadas del último pixel que haya sido PUNTEADO. Las direcciones son:

23677 coordenada de la fila

23678 coordenada de la columna

Por lo tanto, si acabas de usar

PLOT 47, 124

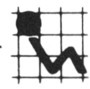
en la dirección 23677 habrá 47, y en la 23678 habrá 124. Comprueba esto:

```
10 INPUT fila,columna
20 PLOT fila,columna
30 PRINT PEEK 23677,PEEK 23678
```

Puedes usar COORDS, obviamente, para determinar dónde estás en un momento dado antes de TRAZAR (que te lleva desde la posición corriente, los desplazamientos especificados en el comando DRAW; como bien sabes). Si quieres tirar una línea hasta un determinado punto a, b -y has olvidado, o no has guardado en algún sitio, la posición corriente- puedes usar

DRAW a - PEEK 23677, b - PEEK 23678

El uso primordial de esta instrucción sería al trazar curvas, pero también puede que la encuentres útil si usas el teclado para puntear a discreción los pixeles (como en el capítulo 1 el programa Dibujero) y deseas saber dónde te encuentras.



¿Alguna vez conseguiste un diagrama realmente bonito en la pantalla, pero con una combinación de colores horrorosa? Y la única manera de cambiarlo era modificando el programa, lo que borra la imagen, por lo que tenías que comenzar de nuevo...

Hay una manera mejor.

7 Fichero de atributos y grafismos

Si fuiste propietario de un ZX81, sabrás que procesa a dos velocidades, llamadas sensatamente rápida (FAST) y lenta (SLOW) y que la pantalla se queda en blanco cuando está operando en modo rápido. Además ya eres consciente (ver Código Máquina y Mejor BASIC) que la imagen de la pantalla es fiel reflejo del contenido de una zona de memoria denominada "fichero de grafismos", que varía dinámicamente y cuya dirección de comienzo es el valor de la variable sistemal denominada D-FILE. (Si no compraste o usaste un ZX81, saltate este párrafo. Lo siento, es demasiado tarde).

El Spectrum no es así: es puro, y sin variable D-FILE.

Los circuitos para la imagen en pantalla, funcionan todo el tiempo, así que no hay velocidad lenta (y por tanto, no se necesitan comandos de velocidad). La información correspondiente a la imagen de pantalla, está contenida en dos recintos de memoria, el fichero de atributos y el fichero de grafismos (vulgarmente lo llamo pizarra). Residen en posiciones prefijadas de la memoria, y funcionan en maneras diferentes.

ATRIBUTOS

El fichero de atributos es más fácil de comprender, y, a no ser que seas un entusiasta del Código Máquina, mucho más útil en todos los sentidos.

La imagen de la pantalla consta de 22 renglones (24 incluyendo el trozo inferior para mensajes) de 32 columnas cada uno, lo que hace un total de 22 * 32 = 704 (o 24 * 32 = 768) posiciones. El comando PRINT AT r, c coloca el cursor en la columna c-ésima del renglón r-ésimo (contando a partir de Ø y hacia arriba). El carácter que se expone en esa posición, está guardado en el fichero de grafismos; pero sus atributos (color, parpadeo, etc.) se guardan en el fichero de atributos. El orden en que los atributos están guardados es directo: se comienza por la parte superior izquierda y se van leyendo los renglones de izquierda a derecha, como si fuera un libro cualquiera. El fichero de atributos comienza en la dirección 22528; así que el atributo para la posición relativa al renglón r, columna c está guardado en la dirección:

    22528 + 32 * r + c

Puedes cambiar el valor en esta dirección, y por tanto los atributos, sin tener que limpiar la pantalla. Es útil, porque los comandos de papel y tinta solamente afectan a lo que se exponga después de haberlos dado.



Cada atributo ocupa un solo octeto, y sus ocho bits se dividen así:

PARPADEO sí/no	BRILLO sí/no	tres bits	para	color papel	tres bits	para	color tinta

en que como es habitual "sí" es 1, y "no" es Ø.

En otras palabras, si el PARPADEO tiene valor f, el BRILLO valor b, el PAPEL valor p, y la TINTA valor i, el valor correspondiente a esos atributos es:

```
128 * f + 64 * b + 8 * p + i
```

Además, puedes determinar los atributos de la posición correspondiente al renglón r, columna c, usando el comando

```
LET att = ATTR (r, c)
```

y obtenido este valor, puedes sacar la lista de valores f, b, p, i usando:

```
LET f = INT (att/128)
LET b = INT (att/64) - 2 * f
LET p = INT (att/8) - 16 * f - 8 * b
LET i = att - 8 * INT (att/8)
```

El siguiente programa convierte la imagen en pantalla para que cada "picto" tenga el mismo atributo (elegido). Por ejemplo, si laboriosamente has construido un mapa-mundi con tinta negra sobre papel blanco y ahora quieres tinta roja sobre papel amarillo, puedes hacer el cambio sin perder el mapa ni comenzar de nuevo. Usa comandos directos, así que teclea:

```
LET f = Ø: LET b = Ø: LET p = 6: LET i = 2
LET att = 128 * f + 64 * b + 8 * p + i: FOR t = Ø
TO 7Ø3: POKE 22528 + t, att: NEXT t
```

Mejor todavía, es tenerlo previamente en memoria, como parte de un programa utensilio general: y luego puedes imponer f, b, p, i. Desde luego, también puedes deducir de cabeza que en este caso att = 8 * 6 + 2 = 5Ø; por lo que el comando podía ser:

```
FOR t = Ø TO 7Ø3: POKE 22528 + t, 5Ø: NEXT t
```

y ya has conseguido el resultado apetecido.

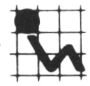
Si cambias el bucle a

```
FOR t = Ø TO 767 etc.
```

también cambiarás además la parte inferior de la pantalla; y

```
FOR t = 7Ø4 TO 767
```

cambia **justamente** ese trozo. La vez que lo necesitas es cuando has cargado una **pintura** (LOAD "Mona Lisa" SCREEN$) y termina con el color equivocado en la parte inferior de la pantalla, debido a que ha habido cambios en el borde desde el momento que guardaste la pintura en cinta. Con esto no ocurre como con BORDCR, capítulo 6, que tienes que poner a cero las variables. No necesitarás esto muy a menudo, pero puede ser simplemente útil.



Para cambios rápidos de atributos, esta rutina es demasiado lenta. El remedio es pasar a Código Máquina, véase Código Máquina del Spectrum por Ian Stewart y Robin Jones, Shiva, capítulo 14.

IMAGEN

Es mucho más complicado, porque cada carácter en la imagen se almacena como una lista de 8 octetos (sus líneas en un picto de 8 x 8 pixeles; véase capítulo 15). Y **no se almacenan en el orden obvio y natural.**

Si alguna vez has cargado una imagen usando SCREEN$, habrás observado el orden peculiar y curioso en que el ordenador "pinta" la imagen. Lo hace exactamente en el orden en que los octetos están almacenados en el fichero de grafismos.

Para ver esto claramente, pasa este pequeño programa:

```
10 INPUT fila, columna
20 PLOT fila, columna
30 PRINT PEEK 23677, PEEK 23678
```

Luego, lo guardas: SAVE "blanco" SCREEN$. Finalmente, CLS y LOAD "blanco" SCREEN$, y vigila lo que sucede. (Si te olvidas de limpiar pantalla, no disfrutarás mucho...)

Es más fácil describir el proceso en función de las coordenadas usadas para los gráficos de alta resolución, una retícula de 256 x 176. Numerando los renglones desde 175 en la parte de arriba hasta 0 en la parte de abajo, y las columnas desde 0 a la izquierda hasta 255 a la derecha.

El fichero de grafismos comienza en la dirección 16384.

Los primeros 32 octetos contienen los valores para la línea 175. Cada octeto gobierna un segmento de 8 columnas en esa línea. El primer octeto trata con las columnas 0-7, el siguiente con las 8-15, y así sucesivamente. Si limpias pantalla y luego tecleas

```
POKE 16384, BIN 01010101
```

o su equivalente decimal

```
POKE 16384, 85
```

verás un trazo de cuatro motas

. . . .

en la parte superior izquierda. Cambia a POKE 16385, 85 y sucesivos, y verás cómo se desplaza el trazo de motas. Las motas, desde luego, son el número binario 01010101 convertido a gráficos: 1 siendo "pinta un pixel" y 0 "borra un pixel" de la retícula de alta resolución.

Cada línea horizontal de esta imagen, está reflejada en el fichero de grafismos como una serie consecutiva y ordenada de 32 octetos.

Desafortunadamente, las líneas por sí mismas no siguen el orden numérico, como ya hemos visto.



Primeramente, van estas líneas:

175 167 159 151 143 135 127 119

(consideralas como la línea superior de las primeras ocho filas de caracteres). Luego el ordenador pasa a las segundas líneas:

174 166 158 15∅ 142 134 126 118

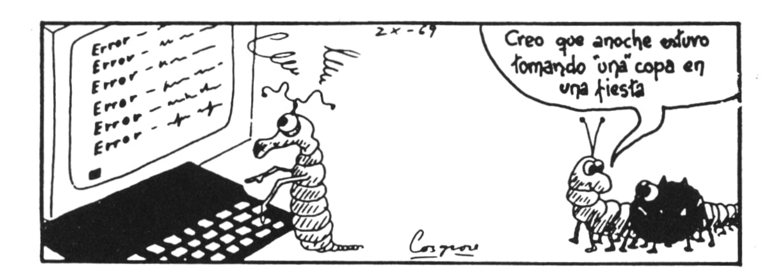
Luego

173 165 157 149 141 133 125 117

y así hasta que llega a la línea 112. En ese momento se ha tratado con el tercio superior de la pantalla (renglones ∅-7 en baja resolución).

Luego, y de forma similar, se trata con el segundo tercio; y finalmente con el tercio del fondo, incluyendo el trozo de pantalla reservado a los mensajes.

Mayormente, te he descrito esto para satisfacer tu curiosidad. Solamente lo necesitas saber si quieres escribir figuras que se mueven en Código Máquina o esa suerte de cosas. Si tú **sí** quieres hacerlo, nos llevaría demasiado espacio describir cómo: **Código Máquina del Spectrum** profundiza en este tema mucho más.

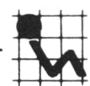


Proyectos

1. Modifica el programa que cambia los atributos, de manera que las diferentes secciones de la pantalla tengan diferentes atributos, usando instrucciones DATA adecuadas. Usalo para producir un mapa-mundi "sólido", como el del capítulo 2 pero con los continentes (en lo que sea practicable) sombreados en diferentes colores, y los océanos en cyano.

2. Si tienes mente matemática, muestra que la siguiente secuencia de instrucciones calcula la posición dentro del fichero de grafismos que corresponde a un pixel que en alta resolución ocupa el renglón r, columna c.

```
10 INPUT r,c
20 LET r1=175-r
30 LET b=INT (r1/64)
40 LET c1=INT (c/8)
50 LET c2=c-8*c1
60 LET r2=INT (r1/8)
70 LET r3=r1-8*r2
80 LET a=2048*b+256*r3+32*r2+c1+16384
90 LET bitpos=c2+1
```



```
100 PRINT 'El pixel con coordenadas□';r;'□';c,
    'se almacena en el fichero de''grafismos
    en la direccion□';a;''como bit□';bitpos;
    '□del octeto'
```

(La posición de cada bit dentro del octeto, va de Ø a 7 a partir del más significativo y hacia el menos significativo, en la forma: Ø1234567).

Sinclair, es de suponer, tuvo una buena razón para usar este orden tan particular...



Y ahora el Spectrum se postra en la cama del psiquiatra para echar una ojeada a ***tus variables sistemales...***

8 Psicoespectrología

...El arte de experimentos psicológicos con un Spectrum.

La idea de este capítulo es usar el ordenador para investigar ciertos fenómenos curiosos en la psicología de la percepción visual: cómo vemos las cosas. El ordenador es ideal para esto: puede establecer imagenes cuidadosamente controladas ("estímulos", como dicen los psicólogos) para destacar rasgos específicos del mecanismo de percepción. (¿Lo ves?, Ya he echado mano de la jerga, y apenas hemos comenzado).

POSTIMAGENES

Si miras a un objeto brillante durante un cierto tiempo, las células del ojo que reciben la luz, se "cansan" y dejan de responder vivamente al objeto. Da como resultado la formación de "postimagenes": motas oscuras con la misma forma que el objeto brillante original. El objeto se desvanece después de unos pocos segundos.

Supón que el objeto original no sólo es brillante, sino **de colores.** ¿Cómo se comportan estas postimagenes? El siguiente programa te permite determinarlo, usando el Spectrum para generar y mantener el estímulo original.

```
10 PRINT AT 19,0;"Postimagenes"
20 PRINT "Mira al cuadrado"
30 INPUT "Color?",p
40 FOR i=1 TO 6
50 PRINT  BRIGHT 1; PAPER p;AT 8+i,13;"□□□□"
60 NEXT i
70 PAUSE 500
80 CLS
90 GO TO 10
```

Cuando rulas este programa, teclea el color pulsando el número correspondiente de la fila superior de teclas. Aparecerá un cuadrado brillante: obsérvalo, intentando no quitar los ojos de él. Cuando desaparece, debieras ver un cuadrado fantasma y borroso, que se desvanece en unos segundos. (Si no es así, o es muy tenue, intenta aumentar la pausa a aproximadamente 1000).

Si tus ojos funcionan como los míos, los colores que ves debieran ser aproximadamente estos:

Original	**Postimagen**
negro	blanco
azul	amarillo
rojo	cyano



Original	Postimagen
magenta	verde
verde	magenta
cyano	rojo
amarillo	azúl
blanco	negro

Muy bien. Puede que el negro sea gris, y que el rojo un poquito castaño, pero son aproximados.

Observa que los códigos para el original y la postimagen suman 7 (e.g. rojo + cyano = 2 + 5 = 7). Eso significa que el color de la postimagen es **complementario** del original: contiene precisamente aquellos tintes de color que no están presentes en el original.

¿Por qué? Probablemente, porque el exceso de exposición al color original, ha reducido la capacidad de las células del ojo para responder a él, lo que aparece como una forma de respuesta exagerada ante los colores complementarios.

En circunstancias ordinarias, no notarás mucho las postimagenes (a menos que observes muy estrechamente el Sol, lo que es peligroso si lo haces por más de una pizca de segundo, así que **no lo hagas**). Esto ocurre porque el ojo está constantemente bailando de un punto a otro. Pero este experimento con el Spectrum las muestra de forma meridiana (no mires **demasiado** tiempo a la pantalla del Spectrum, tampoco!).

LA ILUSION HERING

Data de 1861, y recibe el nombre de su descubridor Ewald Hering.

```
 10 FOR i=5 TO 255 STEP 12
 20 PLOT i,0: DRAW 255-2*i,175
 30 NEXT i
 50 FOR i=5 TO 175 STEP 12
 60 PLOT 0,i: DRAW 255,175-2*i
 70 NEXT i
100 INPUT 'Espaciando?',q
110 PLOT 0,87-q: DRAW 255,0
120 PLOT 0,87+q: DRAW 255,0
```

Traza un manojo de rayos a través del centro de la pantalla, luego pide un dato. Intenta primeramente entre 1Ø y 2Ø. Aparecen dos líneas. Parecen **curvas** (aunque la curvatura de la pantalla puede arruinar un poquito la ilusión).

Sin embargo, queda claro de las instrucciones 11Ø y 12Ø del programa que deben ser **rectas**. Las otras líneas atontan al ojo.

Ensayalo con diferentes valores de espaciado, y diferentes combinaciones de tinta y papel. ¿Qué valores dan la mejor ilusión?



LA ILUSION WUNDT

No fue hasta 1896 en que alguien ensayó lo obvio, e hizo las líneas curvarse de otra manera. El genio responsable fue Wilhelm Wundt, el primer hombre en sugerir que los psicólogos se preocuparan de realizar **experimentos.** Dibujar arbolitos y todo eso...

```
 10 FOR i=-125 TO 125 STEP 10
 20 PLOT 127,0: DRAW i,87: DRAW -i,88
 30 NEXT i
 40 FOR i=2 TO 87 STEP 10
 50 PLOT 0,i: DRAW 127,-i: DRAW 128,i
 60 PLOT 0,175-i: DRAW 127,i: DRAW 128,-i
 70 NEXT i
100 INPUT 'Espaciando?',q
210 PLOT 0,87-q: DRAW 255,0
220 PLOT 0,87+q: DRAW 255,0
```

Funciona de la misma manera. Reconozco que la ilusión es más eficaz en una pantalla de TV que la versión de Hering: ¿tú que piensas?

LA ILUSION POGGENDORF

¿Te has dado cuenta de que todos son alemanes? Johann Poggendorf propuso ésta en 186∅.

```
10 FOR i=0 TO 20
20 PLOT 20,70+i: DRAW 200,0
30 NEXT i
40 PAUSE 50
50 PLOT 20,10: DRAW 200,130
60 IF INKEY$='' THEN  GO TO 60
70 OVER 1: PLOT 113,70: DRAW 30,20
80 OVER 0
```

Este programa pinta un bloque rectangular y dos líneas que radian de él. Parece como si estuvieran desplazadas y no fueran una sola.

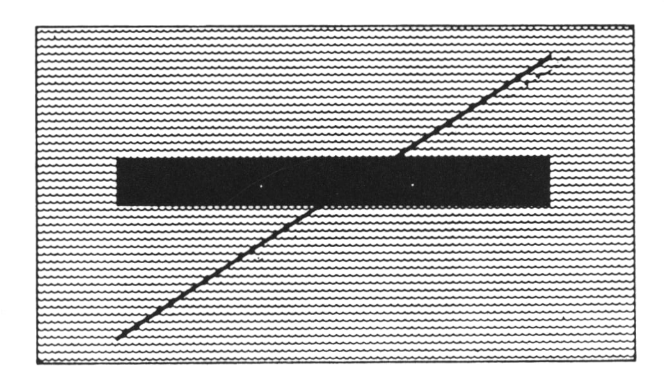


Figura 8.1 La Ilusión Poggendorf: ¿está toda la línea derecha?



Sin embargo, si pulsas una tecla, se dibujará un trazo en blanco entre ellas para mostrar que están muy bien alineadas una con otra: y ahora toda la línea aparece derecha.

Hmmm...

LA ILUSION MÜLLER-LYER

Al principio pensé que eran **dos** alemanes... pero no: Franz Müller-Lyer tiene un apellido doble (Doppelname, como dicen ellos).

```
 10 PLOT 40,130: DRAW 180,0
 20 PLOT 40,60: DRAW 180,0
 30 PLOT 25,45: DRAW 15,15: DRAW -15,15
 40 PLOT 235,45: DRAW -15,15: DRAW 15,15
 50 PLOT 55,115: DRAW -15,15: DRAW 15,15
 60 PLOT 205,115: DRAW 15,15: DRAW -15,15
 70 IF INKEY$='' THEN  GO TO 70
 80 FOR t=1 TO 7
 90 PLOT 40,140-10*t: DRAW 0,-7
100 PLOT 220,140-10*t: DRAW 0,-7
110 NEXT t
```

Ya has visto esta. Son dos flechas con las cabezas apuntando en diferente dirección. La inferior es claramente más larga. ¿O no? Pulsa cualquier tecla: mira las líneas de puntos...

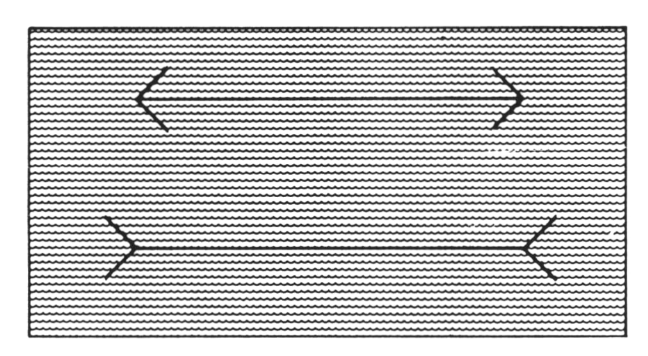


Figura 8.2 La Ilusión Müller-Lyer: ¿son de la misma longitud?

LA ILUSION WERTHEIMER

Debida a Max Wertheimen en 1912...

```
10 PAPER 0: INK 6: BORDER 0: CLS
20 INPUT 'Pausa?',p
25 INPUT 'Numero?',n
30 FOR t=1 TO 20
35 FOR j=1 TO n
40 PRINT AT 8,15+2*(j-5);'□'
45 NEXT j
```




```
 50 PAUSE p
 55 FOR j=1 TO n
 60 PRINT AT 8,15+2*(j-5);'■'
 70 PRINT AT 14,15+2*(j-5);'□'
 75 NEXT j
 80 PAUSE p
 85 FOR j=1 TO n
 90 PRINT AT 14,15+2*(j-5);'■'
 95 NEXT j
100 NEXT t
```

Cuando lo rules, te pedirá un valor para la pausa -algo entre 1 y 1ØØ está bien- y un número. Te sugiero que impongas uno en los primeros ensayos.

La imagen muestra un cuadrado amarillo (o dos). Si la pausa es pequeña, verás dos luces (posiblemente parpadeando) si la pausa es grande, verás una luz durante un rato, luego la otra. Pero si la pausa es intermedia, lo que ves es una **luz**, botando arriba y abajo (ensaya con PAUSA = 35). Puedes ver este efecto con las luces antiniebla en las carreteras británicas.

Si ensayas con números mayores de 1 (y no más de 1Ø ó 12) verás una entera ristra de luces. Lo que tú percibes, depende un poco del hecho que no están parpadeando, completamente sincronizadas: el programa BASIC tarda tiempo en efectuar el bucle. Experimenta.

Esta ilusión es importante para los juegos de ordenador: los gráficos movientes no funcionarían sin él. A propósito, ni lo haría la televisión ni las películas. Ni el Oso Yogi ni los Pitufos.

LA ILUSION SCHWINDEL

...Debida a Hans-Wilhelm Schwindel en 1872. Bien, realmente no: la inventé yo mismo, pero nunca lo hubieras adivinado, a menos que supieras que en alemán "Schwindel" corresponde a "impostor o farsante". Sin ninguna duda, media docena de alemanes lo inventaron hace medio siglo, pero no sobre un Spectrum.

```
10 OVER 1
20 POKE 23607,50
30 FOR i=1 TO 704: PRINT CHR$ (35+10*RND);: NEXT i
40 POKE 23607,60
50 LET i=125*RND: LET j=85*RND
60 LET i1=125*RND: LET j1=85*RND
70 PLOT i,j: DRAW i1,j1
80 PAUSE 20
90 GO TO 50
```

Este dibuja aleatoriamente una imagen llena de motas en alta resolución. (No lo interrumpas, u obtendrás mensajes extraños. Si **sí** interrumpes, teclea POKE 236Ø7, 6Ø).



Luego comienza dibujando líneas. Tú ves que las líneas entran, pero una vez que están dentro, pierdes totalmente el rastro de dónde están. El ojo está detectando **cambios** aleatorios en la imagen, pero no puede retener la imagen resultante con ningún detalle. Es algo que tiene que ver con la manera en que percibimos la **textura** de los tejidos.

Para una variante, ensaya con círculos. Cambia las instrucciones 5Ø-7Ø de manera que digan:

```
50 LET i=50+150*RND: LET j=40+90*RND
60 LET r=20*RND+10
70 CIRCLE i,j,r
```

De nuevo, ves cómo aparecen; pero la imagen no dura. Si aún no estás convencido de que los círculos entran completamente, añade un cambio de color:

```
55 INK 3
```

Ahora verás que algo está cambiando; pero no comienzas a ver los círculos hasta que una gran parte de la pantalla se ha puesto magenta. El cambio de color parece impresionar el ojo (y el cerebro) incluso más que el cambio en los pixeles individuales. Y oscurece este último cambio casi completamente.



En su mayoría, el uso comercial de ordenadores requiere enormes cantidades de información. Se almacena en discos o en cintas en forma de...

9 Ficheros

¿Qué es un fichero, en el contexto de cómputo? Los científicos tienen el desconcertante hábito de adoptar una palabra de uso común y darle un significado sutilmente alterado para que se acomode a sus propósitos. Los científicos de ordenadores han hecho justamente eso con la palabra **fichero**. Para ellos, la palabra implica una enorme (habitualmente) colección de datos que guardan relación con algún tema específico.

Así por ejemplo, un agente inmobiliario tendrá registrados en sus ficheros los detalles de las propiedades y parcelas que tiene disponibles para la venta. Fácilmente puede decirle a su secretaria, "tráeme la ficha de la Avda. Acacia, 36, por favor"; pero debe tener cuidado si hay algún ingeniero de computación, porque puede decirle con aires de suficiencia que no utiliza correctamente la palabra, dado que lo que realmente quiere es un **registro** del fichero. Nuestro agente puede argüir, sin que haga cambiar de opinión al ingeniero, que él ya usaba la palabra "ficha", y llamaba a las colecciones de fichas "ficheros", de forma obvia, antes de que el hombre pensara en ordenadores. El ingeniero de computación seguirá ignorando sus argumentos y camuflará el hecho de no contestarlos haciendo notar que cualquier detalle de ese registro, como por ejemplo, el propietario actual de la parcela en la Avenida de Acacia, 36, o el precio que se pide por ella, se denomina **campo**. En este momento, el agente inmobiliario puede que arguya que en su ficha ese era el dato correspondiente a una casilla, y que campo era precisamente lo que la inmobiliaria había urbanizado para hacer parcelas; o bien se olvidará de la discusión.

Echémos una ojeada a otro ejemplo de un fichero de naturaleza más personal. Supongamos que quieres mantener detalladamente los discos de tu colección. El fichero es la suma total de todos los datos sobre esta colección. Cada ficha -registro- del fichero es obviamente, la información sobre un disco (de los que ruedan a 33⅓ r.p.m.) de la colección, y puede estar en un disco (de los que permiten grabar magnéticamente información) o en una cinta en cassette. Supongamos, por simplicidad, que todos los registros son de música pop, y por tanto, tienen 12 canciones en secciones separadas de ambas caras. La pista en la que pincha la aguja es sólo una, pero por eso de cambiar sutilmente los nombres, también podemos decir que tiene 12 pistas. De forma que cada ficha -cada registro- del fichero, consta de los campos -de las casillas- dadas en la tabla 9.1.



Tabla 9.1

Campo Nº	Descripción	Nº de caracteres (máx.)
1	Artista	3Ø
2	Fecha de compra	6
3	Título pista 1	2Ø
4	Duración pista 1 (minutos)	5
5	Título pista 2	2Ø
6	Duración pista 2 (minutos)	5
7	:	
8	:	
.	y así sucesivamente	
.	:	
.	:	
.	:	
25	Título pista 12	2Ø
26	Duración pista 12 (minutos)	5
	Nº total de caracteres por registro:	336

Cada uno de los campos descritos en la tabla, tiene una longitud prefijada. Así que, por ejemplo, si el artista es Juan Pardo -que sólo ocupa 1Ø posiciones (incluyendo el espacio entre palabras), se añadirán 2Ø espacios más para completar los 3Ø símbolos standard. Este "3Ø" es una estimación razonable en cuanto al número máximo de posiciones que el nombre del Artista puede ocupar. Debiera ser adecuado para la mayoría de los propósitos; incluso "Bonzo Dog Doo-Dah Band" cabe, pero no "Dave Dee, Dozy, Beaky, Mick and Titch".

Se puede objetar ahora, que se va a desperdiciar de esta manera un montón de memoria, y que sería mejor, permitir que los campos tuvieran longitud variable, y delimitar cada uno de ellos con algún símbolo especial. A eso, sólo puedo contestar que tu argumento es sensato, pero hacer cosas como esa, haría mucho más difícil la programación de lo que estoy preparado para conseguir; así que quedan con longitud prefijada y constante. En cualquier caso, y en algunos otros pocos casos, no hay cuestión sobre la longitud del campo. Por ejemplo, el campo "Fecha de compra" siempre contiene exactamente 6 caracteres: 2 para el día, 2 para el mes y 2 para el año; como en Ø31182 para el día 3 de noviembre de 1982. Similarmente, la duración de una canción -de una pista- está **casi** prefijada. He admitido dos posiciones decimales, así que por ejemplo, la duración puede mencionarse como 3,16 minutos. Eso sólo ocupa cuatro posiciones, (incluyendo la coma decimal), y yo le he dado cinco. Es simplemente como precaución ante la posibilidad de que una canción dure diez minutos (o más).

Como te muestro, el registro -la ficha- completa, ocupa 336 posiciones usando estas hipótesis. Así que si tenemos 2ØØ discos en nuestra colección, el fichero va a ocupar 672ØØ octetos, lo que es bastante más de lo que el Spectrum tiene posibilidad en su memoria, en una sola tacada!.

Ese es un rasgo típico de los fichero informáticos; siempre observamos que con mucho, ocupan más espacio del disponible en la memoria principal. Eso significa que tenemos que apoyarnos de memoria auxiliar, y para nuestros propósitos, eso significa cintas en cassette.



FICHEROS EN CINTA

¿Qué forma debiera adoptar el fichero en cinta?

Antes de contestar esa pregunta, sería provechoso pensar un poco en cómo vamos a usarlo cuando ya lo tengamos. Una cosa que va a ser necesaria con bastante frecuencia es revisar el fichero para incluir los nuevos discos que compremos, y para suprimir de él los que hemos desechado. Es lo que se denomina como **mantenimiento del fichero.**

Ahora bien, **realmente** no podemos quitar nada de un pequeño trozo de cinta. Porque quedaría el lugar en blanco (a no ser que lo cortáramos y volviéramos a pegar la cinta). La única manera de hacer la tarea es copiar todo el fichero, excepto por el trozo que queremos suprimir, desde una cinta a otra. La cinta original queda invariada, pero la nueva ya no contendrá el registro suprimido. La figura 9.1 indica cómo se puede disponer esto, por lo menos en principio.

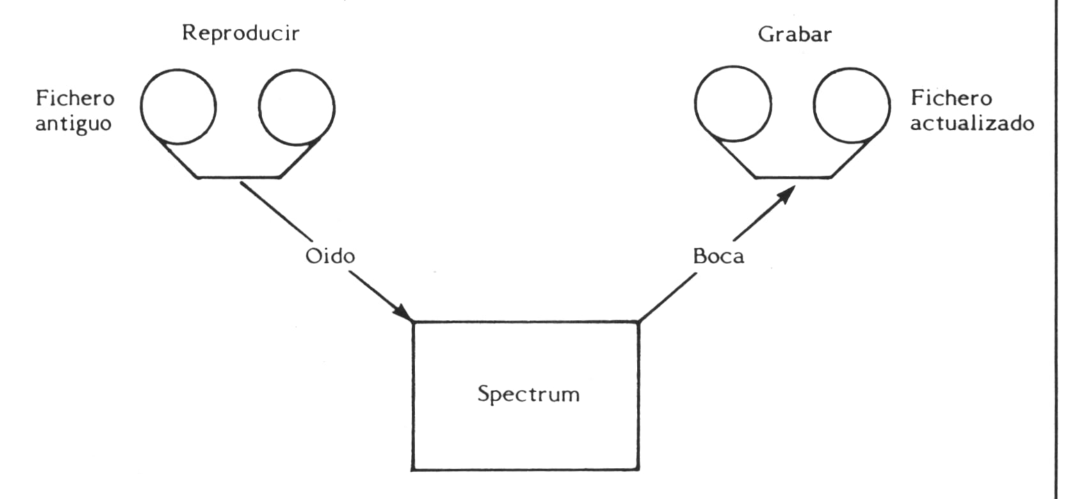


Figura 9.1 Uso de dos cassettes para manejar ficheros conservados en cinta.

Así que necesitamos dos cassettes, una en el modo "reproduciendo" (conectada al enchufe "oido"), y la otra en modo "grabando" (conectada al enchufe "boca" o micrófono).

¿Cuáles son las consideraciones para programar? En esquema, la codificación es:

1. Leer un registro.
2. Si es el registro final, entonces acabar.
3. Si es el que queremos suprimir, volver a la tarea 1.
4. Escribir un registro.
5. Ir a 1.

Ahora bien, si dejamos la cinta corriendo, no hay garantía de que cuando hayamos efectuado las tareas 2, 3 y 4, no se haya pasado ya el siguiente registro. Así que obviamente, cada vez que leemos un registro, tendremos que detener la cinta.



Igualmente, no tiene ningún sentido dejar corriendo la cinta que está grabando cuando no tenemos nada que escribir en ella. Dado que nos ocupará más de dos segundos, leer o escribir un registro, la puesta en marcha y la detención del motor de arrastre muy rápidamente de cinta, va a resultar bastante laboriosa.

BLOQUES DE DATOS

Así que lo que necesitamos es un compromiso entre acción -lectura y escritura- y paradas. Como no podemos conservar todo el fichero en memoria y tampoco queremos tratarlo registro a registro, lo que debemos es separarlo en bloques de un cierto número de registros cada uno, de manera que podamos conservar cómodamente en memoria un bloque de una vez. Para un Spectrum de 16K, hay disponibles para el usuario 9K. Si el programa nos ocupa 1K, podemos permitirnos bloques de aproximadamente 4K para tener, en cualquier momento dado, un bloque de 4K que ha sido metido a la memoria (al **buzón de entrada***) y cuyos registros se están transfiriendo al **buzón de salida*** para ser escritos subsecuentemente a la cinta. Ahora nuestra organización es la de la figura 9.2.

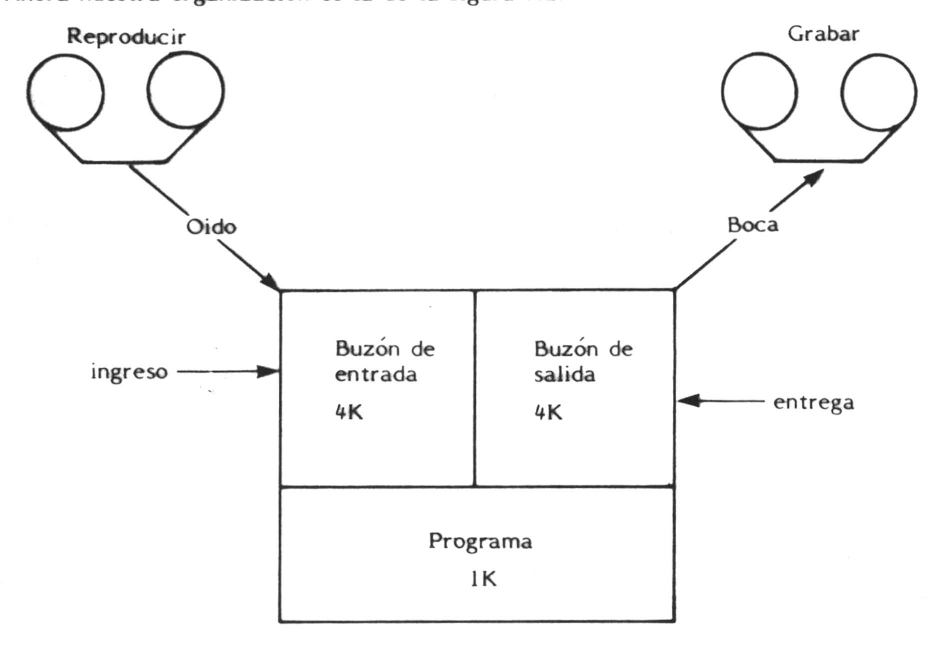


Figura 9.2 Disposición en memoria de las zonas para manejo de ficheros

Para el ejemplo de la colección de discos, seremos capaces de conseguir 12 registros por bloque.

En lo que concierne al usuario, será conveniente si él no tiene que pensar sobre los aspectos **domésticos** de esta disposición. De hecho, sería más simple si todo **pareciera** como si leyeramos y escribieramos registro a registro. Necesitaremos dos subrutinas para esto "coger un registro" (leer) y "dejar un registro" (escribir). La mayoría del tiempo, estas rutinas no estarán haciendo realmente ninguna lectura ni escritura, sino simplemente transfiriendo datos desde el buzón de entrada o hasta el buzón de salida.

* Input buffer/Output buffer



Sin embargo, cuando se vacíe el buzón de entrada, será necesario leer el siguiente bloque; y al contrario, cuando se llene el buzón de salida, habrá que escribir ese bloque. Llamamos a esas rutinas **cogebloque** y **dejabloque.**

Hay un par de consideraciones más: primeramente necesitamos dos **punteros** (pin y pex), para mostrar lo llenos que están los buzones en un momento dado. Para comenzar, tendremos que ponerlos a cero; así que tendremos, para hacer esto, una rutina denominada **iniciar** y para establecer cualquier otra condición inicial que resulte necesaria. En segundo lugar, no hemos pensado sobre cómo debemos terminar el fichero. Obviamente, no hay ninguna garantía de que el fichero sea exactamente un número de bloques completos, así que tendremos que plantear alguna forma de forzar la activación de **dejabloque** al reconocer alguna clase de delimitador de fichero. Adoptaremos el convenio que el campo 1 del último registro contiene "ultimal" (sobre la base que es muy poco probable, que sea el nombre de ningún artista de la canción), y que los otros campos de ese registro no significan nada, de forma que en la práctica pueden contener cualquier cosa.

Otra cosa más. No hemos dicho exactamente cómo vamos a estipular los buzones de entrada y salida. Claramente, ambos son cadenas de caracteres. Permitiremos al usuario la posibilidad de decidir cuán grandes han de ser los buzones para su aplicación particular. La forma más simple de hacer esto, es preguntarle el número de registros por bloque (nrb) y la longitud de cada registro (lpr). Luego establecemos dos tablas "litéricas" de dimensiones:

DIM i$ (nrb, lpr) [buzón de entrada]

DIM e$ (nrb, lpr) [buzón de salida]

Lo que puede hacerse en la rutina **iniciar.**

Pertrechados con estas ideas, podemos comenzar a mirar la forma en que trabajarán las rutinas **pinza** y **punza.** En esquema, son como sigue.

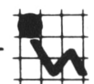
Pinza

1. IF pin = Ø o pin > nrb THEN cogebloque
2. IF i$ (pin) = "ultimal" THEN PRINT "Llegando al registro ULTIMAL": STOP
3. Transferencia de i$ (pin) hasta r$
4. Incremento de pin
5. REGRESO

El movimiento del puntero pin necesita un poquito de explicación. Para comenzar, **iniciar** lo pondrá a cero, lo que significa desde luego, que el buzón está vacío. Si ahora se hace una llamada a **pinza,** tendrémos que recurrir a **cogebloque.** De eso nos preocupamos en la instrucción 1, y en **cogebloque** pondremos pin a 1 para indicar dónde está el primer registro a leer; y éste será pasado a continuación a r$ (línea 3) para que pueda usarlo el programa llamante. Ahora incrementaremos pin de forma que en la siguiente llamada a **pinza,** sea el segundo registro el que se transfiere a r$. Todo eso está bien hasta que todos los registros de ese bloque hayan sido transferidos, momento en el que pin estará señalando más allá del extremo de la tabla (i.e. pin es mayor que nrb). Por eso es por lo que escribimos las dos condiciones de la línea 1 antes de llamar a **cogebloque.**

Punza

1. Incremento de pex
2. Transferir r$ a e$ (pex)
3. IF pex = nrb OR r$ = "ultimal" THEN dejabloque



4. REGRESO

La línea 1 incrementa el puntero pex directamente, porque **iniciar** lo ha puesto a cero, y eso ha sido antes de empezar a funcionar con la tabla de salida e$. Así que, en la primera llamada, lo que está en r$ será transferido a e$ (1) que es lo que queremos. Sabemos que el buzón está lleno si pex = nrb, pero además, queremos forzar la ejecución de **dejabloque** si hemos llegado a nuestra marca de final de fichero. Por eso es por lo que ambas condiciones se comprueban en la línea 3. A propósito, **dejabloque** tiene que reponer pex al valor cero, de forma que vuelva a ponerse a 1 al comienzo de la primera llamada a punza después de una llamada a **dejabloque.**

Las rutinas **cogebloque** y **dejabloque,** son bastante directas también, pero tenemos que solventar un problema más antes de bosquejarlas: no hemos pensado sobre cómo denominar a los datos que vamos a guardar en cinta. Podemos dar a cada bloque el mismo nombre, pero fácilmente produciría confusión y la confusión implica problemas. Una técnica mejor es permitir al usuario que dote de nombres a sus ficheros (otra vez dentro de **iniciar**) y luego altere automáticamente el nombre dentro de **dejabloque,** de manera que cada bloque tenga un número diferente. Por ejemplo, si el usuario denomina al fichero "tocata", los bloques se guardarán realmente como tocataØ, tocata1, tocata2, etc. Similarmente, **cogebloque** tendrá que llevar la cuenta de los bloques al ingresar en el buzón. De nuevo, la cuenta de bloques será labor de la rutina **iniciar.**

Así que el bosquejo de las rutinas es:

cogebloque

```
PRINT "marchando un PINCHADO de cinta"
LOAD fichero de entrada + cuentabloques ingresados DATA i$ ( )
PRINT "amputando. ACABOSE la rascazón"
LET pin = 1
LET cuentabloques ingresados = cuentabloques ingresados + 1
RETURN
```

dejabloque

```
PRINT "marchando un PUNCHADO de cinta"
SAVE fichero de salida + cuentabloques entregados DATA e$ ( )
PRINT "amputando. ACABOSE la grabazón"
LET pex = Ø
LET cuentabloques entregados = cuentabloques entregados + 1
RETURN
```

Nota que cuentabloques ingresados y cuentabloques entregados van a requerir algo más de maña para manejarlos de lo que parece, porque parte del tiempo están usados como "literos" a añadir a los nombres de fichero, y parte del tiempo como "números" que van a ser incrementados. Además, tenemos ahora unas cuantas variables literales desparramadas, que no son fácilmente identificables porque solamente pueden tener nombres con un solo símbolo; por lo que antes de codificar realmente, repasemos la lista de nombres y funciones.



Nombre	Función
i$	buzón de entrada
e$	buzón de salida
r$	registro que aparece al usuario como escrito o leído
f$	nombre del fichero de entrada
g$	nombre del fichero de salida
m$	lítero a empalmar al nombre del fichero de salida, y que corresponde al "número" del bloque que se entrega.

LA CODIFICACION

Comenzarémos el programa para manejo de ficheros en cassette a partir del número 95ØØ, permitiendo 1ØØ para cada rutina, por lo que **iniciar** está en la 95ØØ, **pinza** en la 96ØØ, etc. Así que necesitaremos las líneas 1 a 3:

```
1 LET iniciar=9500: LET pinza=9600
2 LET punza=9700: LET cogebloque=9800
3 LET dejabloque=9900
```

en cualquier programa que use el sistema de control de ficheros en cinta.

Escribamos **iniciar:**

```
9500 LET pin=0: LET pex=0
9510 LET inbc=0: LET exbc=0
9520 INPUT 'Nombre fichero entrada';f$
9525 INPUT 'Nombre fichero salida';g$
9530 INPUT 'Longitud del registro ';lpr
9540 INPUT 'No. de registros por bloque ';nrb
9550 DIM i$(nrb,lpr): DIM e$(nrb,lpr)
9560 RETURN
```

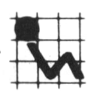
Hasta ahora ningún problema. La rutina **pinza** también es bastante directa:

```
9600 IF pin=0 OR pin>nrb THEN  GO SUB cogebloque
9610 IF i$(pin)( TO 7)='ultimal' THEN
     PRINT 'Llegando al registro   ULTIMAL': STOP
9620 LET r$=i$(pin)
9630 LET pin=pin+1
9640 RETURN
```

y lo mismo sucede con **punza:**

```
9700 LET pex=pex+1
9710 LET e$(pex)=r$
9720 IF pex=nrb OR r$='ultimal' THEN  GO SUB dejabloque
9730 RETURN
```

Ambas son bastante idénticas a los programas bosquejados, ¿o no?
Las rutinas **cogebloque** y **dejabloque**, requieren un poquito de toma y daca malabarista para manejar las conversiones de líteros a números:



```
9800 LET m$=STR$ inbc
9810 PRINT 'Marchando un PINCHADO de cinta'
9820 LOAD f$+m$ DATA i$()
9830 PRINT 'Amputando. Acabose la RASCAZON'
9840 LET pin=1
9850 LET inbc=inbc+1
9860 RETURN
9900 LET m$=STR$ exbc
9910 PRINT 'Marchando un PUNCHADO de cinta'
9920 SAVE g$+m$ DATA e$()
9930 PRINT 'Amputando. Acabose la GRABAZON'
9940 LET pex=0
9950 LET exbc=exbc+1
9960 RETURN
```

COMPROBANDO

Ahora, lo que se necesita es un pequeño programa para comprobar que todo funciona. No queremos tener que teclear una enorme pila de fruslerías, así que estableceremos pequeños buzones y haremos que el programa genere su propio fichero. La cosa más simple es justamente generar una serie de número correlativos.

Debiera hacerse esto:

```
10 GO SUB iniciar
20 FOR n=1 TO 100
30 LET r$=STR$ n
40 GO SUB punza
50 NEXT n
60 LET r$='ultimal'
70 GO SUB punza
80 STOP
```

Pásalo. La primera cosa que sucede es que **iniciar** te pide el nombre del fichero de entrada. Desde luego, no hay ninguno porque no lo hemos creado todavía -eso es lo que nos disponemos a hacer-. Así que dale un nombre cualquiera, "nulo" por ejemplo. Ahora, te pregunta el nombre del fichero de salida, al que puedes llamar "prueba" o "Pinocho", o lo que sea. Luego, **iniciar** te pregunta la longitud del registro en octetos. En este caso, nunca es más de 3 (cuando n = 100) pero ¡ojo al parche! "ultimal" ocupa 7 octetos, así que todos los registros deben tener como mínimo 7 octetos de longitud. (Si no te gusta esto, siempre puedes usar algunos caracteres especiales. vigila y cuidado -los caracteres de control pueden tener efectos extraños!). Finalmente, **iniciar** desea saber el número de registros por bloque. Elige 20, por razones que serán claras y meridianas dentro de un momento.

Ahora, el bicho hace una pizca de tratamiento hasta que llena el buzón de salida, en que, desde luego, se llama a la rutina **dejabloque.** Por lo que tú ves aparecer en la pantalla el mensaje:

"Marchando un PUNCHADO de cinta"

Dado que lo que viene a continuación es el comando para guardar en cinta el fichero, verás el aviso usual para encender la grabadora de cassette y pulsar cualquier tecla. Así que si lo deseas, puedes ahorrarte la línea 9910. La única desventaja haciendolo así es que el aviso standard del Spectrum nos señala que la cinta debe colocarse en el modo de grabación. A propósito, cuando estás manejando un fichero de entrada y uno de salida, parece que se requieren más manos de las naturales.



Las cosas pueden hacerse ligeramente más sencillas dejando las dos cassettes en los modos PLAY y RECORD respectivamente, y controlando sus movimientos usando el botón de pausa (siempre y cuando tus cassettes lo tengan, desde luego).

Mientras tanto, y volviendo al programa, tan pronto como se haya entregado ese bloque, obtienes el aviso para apagar el cassette. En este caso, y a causa de que el buzón se llena muy rápidamente, obtendrás casi inmediatamente otro mensaje para comenzar otra grabación. Así que, apenas merece la pena preocuparse. Todo lo que sucede si no cortas el cassette entre los bloques es que obtienes intervalos ligeramente mayores entre los bloques.

En total, el programa guardará seis bloques, dado que se necesitan exactamente cinco para los números 1 a 1ØØ, lo que indica que el sexto sólo contiene "ultimal".

Ahora necesitamos comprobar que el fichero ha sido por supuesto, guardado correctamente. La cosa más simple a hacer es cambiar las líneas 2Ø, 3Ø y 4Ø para que sean:

```
20 GO SUB pinza
30 PRINT r$
40 GO TO 20
```

y rularlo de nuevo, con el cassette en el modo PLAY. Lo que debería suceder es que después de **iniciar** ha preguntado los detalles del fichero (esta vez es el fichero de salida el que es nulo), el sistema avisa que se apague el cassette, luego la rutina **cogebloque** lee 2Ø números que pasa sucesivamente a la rutina **pinza,** que a su vez los pasa a r$ uno a uno; después de lo cual, son mostrados en pantalla, y se saca un aviso para que se apague el cassette. Y eso es lo que sucede con un par de añadidos. Primeramente, desde luego, el Spectrum muestra el nombre del fichero que está leyendo, así que podremos ver:

character array: tocataØ (o como le hayas llamado)

en la primera pasada.

PROBLEMAS DE DESROLLE

Eso desde luego, es lo que sucede, pero se nos ha colado una mosca en la sopa: el mecanismo de desrolle se mete por medio. El sistema te pregunta si quieres desrollar y cuando respondes afirmativamente, inmediatamente sale con el mensaje:

"Amputando Acabose la RASCAZON"

Así que si eres lento para responder, puedes encontrarte con la cinta a medio camino del siguiente bloque antes de que el programa haya comenzado a leerla. (Este problema es patente particularmente, con un tamaño de bloque de 2Ø registros, que es por lo que lo elegí). No es un desastre total porque el Spectrum sólo intenta leer el bloque que **se supone** que es el siguiente, así que puedes rebovinar un poco; pero eso es bastante tedioso. Una alternativa mejor es desactivar conjuntamente el mecanismo de páginas del Spectrum -el desrolle- durante la actuación de **cogebloque.**

Recuerda del capítulo 6 que hay una variable sistemal llamada SCR-CT con sede en 23692 que puede usarse para esto. Contiene el número de líneas (más 1) que se mostrará en pantalla antes de emitir el siguiente mensaje "scroll?". Así que si hincamos ahí el valor 255 (el máximo posible) siempre que se llame a la rutina **cogebloque,** quitaremos la mosca de la sopa. Así que añadamos:

```
9825 POKE 23692, 255
```



Debe ir después de la instrucción de carga, porque esa instrucción restaura el valor de SCR-CT a 1.

Ahora, cada vez que recurramos a la rutina **cogebloque** nos permitimos sacar en pantalla 256 líneas antes de que aparezca la pregunta de desrrolle. Si esto es adecuado depende de la aplicación. En este caso, es más que suficiente, pero pudiera ser más seguro incluir esa instrucción en la rutina **pinza** así como en la rutina **cogebloque,** dado que se recurre a esa rutina más a menudo. Incluso entonces, el efecto no está garantizado absolutamente, dado que depende de cuanto escriba el programa de usuario en la pantalla antes de la siguiente lectura, pero bajo circunstancias normales no habrá problemas.

SIN NOVEDAD... POR AHORA

Vamos a hacer una pausa para un respiro, y revisar lo que está pasando. Nota primero que **iniciar** se ha utilizado de dos maneras distintas:

1. Para **crear** un fichero sobre el que grabar;
2. Para **abrir** un fichero que ya existe y va a ser subsecuentemente leído.

Así que pudieramos haber escrito dos rutinas separadas en lugar de **iniciar**, llamadas **crear** y **abrir**; y hay algo, desde luego, que tiene que decirse en cuanto a esta aproximación, dado que como hemos visto, si sucede que no queremos un fichero de entrada y uno de salida simultáneamente, tenemos que darle nombres **postizos** para que **iniciar** sea feliz.

En segundo lugar, se le exige al usuario que maneje la terminación de los ficheros por sí mismo. En otras palabras, él tiene que saber que el fichero está delimitado por "ultimal". Podríamos haber escrito otra rutina que llamaríamos **cierre,** que hiciera automáticamente esa tarea, de manera que las líneas 6∅ y 7∅ de nuestro programa de comprobación pudieran sustituirse por:

```
6∅  GO SUB cierre
```

y **cierre** sería justamente:

```
LET r$ = "ultimal"
GO SUB punza
RETURN
```

MICRODUCTORA

Ahora, si observas el teclado del Spectrum, verás como palabras clave: CLOSE # y OPEN #. Son los comandos equivalentes a los que hemos estado comentando, para ficheros en diskettes. (No hay CREATE #, de forma que OPEN # debe hacer esas tareas, igual que las hace **iniciar**). Los equivalentes para **pinza** y **punza** son INPUT # y PRINT #. Todo lo demás (la organización de los bloques del fichero y demás) es manejado por el sistema operativo del Spectrum, y por tanto, la estructura de ficheros en diskette es "transparente" al usuario como lo es la de ficheros en cassette que hemos descrito. Realmente, las rutinas que manejan microductoras de diskette tienen que hacer mucho más que **pinzar, punzar, coger bloques, dejar bloques** e **iniciar**, pero los principios son similares.



CONTROL AUTOMATICO

Hay una pregunta que te habrá estado rondando durante algún tiempo: "¿podemos hacer que el Spectrum controle automáticamente los motores del cassette?".

La respuesta es que sí, y no es muy difícil. Necesitas cassettes que tengan enchufes para control remoto (la mayoría lo tienen). También necesitas un "portal" paralelo de entradas/salidas y un par de relés de 5 voltios. Los contactos de los relés se usan para completar los circuitos de control remoto del motor, y sus bovinas están gobernadas por cualquier pareja de bits del portal. Luego, en lugar de presentar mensajes, simplemente fijamos el portal con ciertos valores en cada bit (usando BIN). El Apéndice B da detalles de los circuitos. Desafortunadamente, SAVE todavía envía automáticamente su aviso para encender el cassette, y espera que se pulse una tecla. Así que, una automatización total está todavía tentándonos por encima de nuestro alcance.

USANDO FICHEROS

Ahora bien, recordarás que los ímpetus para todos estos esfuerzos vinieron de la idea de manejar los detalles de nuestra colección de discos, y parece ser que nos hemos perdido involucrándonos en los detalles del manejo de ficheros en cinta. Pero, desde luego, ahora que ya lo sabemos, nos será mucho más fácil escribir el programa primitivo.

Hay tres funciones básicas en cualquier fichero de sistemas:

1. Crear el fichero a partir de la nada
2. Mantenerlo haciendo las necesarias inserciones y supresiones.
3. Escrutar el fichero en busca de un determinado registro.

Por simplicidad, las escribiremos como programas separados, aunque sería una nimiedad vincularlas juntas por medio de un menú (véase nuestro libro Código Máquina y Mejor Basic).

La rutina de creación de fichero comienza bastante directamente. Habrá una llamada a **iniciar** en que el fichero de entrada se pone a "nulo", el fichero de salida a "reccol" (por ejemplo); la longitud del registro a 336 y el número de registros por bloque, algo así como 5 para permitir al programa un espacio cómodo.

Ahora tropezamos con el único problema serio en esta rutina. Hay que estipular cada registro en una manera conveniente para el usuario. Por ejemplo, sabemos que el campo "Artista" tiene 30 octetos de longitud, pero no queremos que el usuario teclee el nombre de "ABBA" seguido de 26 blancos. Así que tendremos una subrutina llamada **inreg** que maneje el ingreso de un solo registro de manera amistosa y cómoda.

El programa **crear** será entonces como éste:

```
100 GO SUB iniciar
110 GO SUB inreg
120 GO SUB punza
130 INPUT 'Alguno mas (s/n)';q$
140 IF q$='s' THEN  GO TO 110
150 GO SUB cierre    (Suponiendo que lo hayas implementado)
160 STOP
```



Vamos a preocuparnos ahora de **inreg**. Vamos a establecer una tabla litérica, a$, que va a contener el registro a medida que lo construimos. Así que si **inreg** está en la línea 8000:

```
8000  DIM a$ (336)
```

Podemos requerir al usuario el primer dato preciso, y colocarlo en el lugar correcto:

```
8010  INPUT "Artista"; a$ (TO 30)
```

Luego

```
8020  INPUT "Fecha de compra"; a$ (31 TO 36)
```

Después de lo cual, trataremos las 12 pistas de la misma manera:

```
8030  FOR t = 1 TO 12
8040  PRINT "pista"; t
```

Y ahora queremos escribir algo como:

```
8060  INPUT "título"; a$ (principio TO fin)
```

habiendo determinado en la línea 8050 lo que "principio" y "fin" vale.

Obviamente, "principio" y "fin" cambian en función de la pista en la que estemos en este momento. Escribamos una breve tabla de los valores para darnos una idea de las relaciones involucradas.

Pista (t)	Principio	Fin
1	37	56
2	62	81
3	87	106

Así que, principio = 37 + 25 * (t - 1)
y fin = 56 + 25 * (t - 1)

Por lo tanto:

```
8050  LET principio = 37 + 25 * (t - 1): LET fin = 56 + 25 * (t - 1)
```

La duración de la pista va desde principio +1 hasta fin + 5:

```
8070  INPUT "duración"; a$ (fin + 1 TO fin + 5)
```

Y luego:

```
8080  NEXT t
```

Pasamos el resultado a r$:

```
8090  LET r$ = a$
```

y como ya tenemos todo:

```
8100  RETURN
```




MANTENIMIENTO

¿Qué pasa con el programa de mantenimiento? De nuevo, comenzará con una cita a **iniciar**, y por primera vez, queremos definir ambos ficheros, el de entrada y el de salida. El fichero de entrada puede llamarse "colevieja", y para identificar el fichero de salida como actualización de éste de entrada, podremos llamarle "colenueva". Las actualizaciones posteriores podrían denominarse "colenuevb", "colenuevc", y así sucesivamente, a la manera de las matrículas de los coches. O puede que prefieras dar información de la fecha y llamar al fichero, digamos cl982, para indicar colección en Septiembre del 82. En cualquier caso, se necesita **algún** sistema formal, porque de lo contrario, es demasiado probable coger un fichero equivocado y modificar la información incorrecta.

Para comenzar, no nos complicaremos la vida, y haremos que el usuario sólo haga una inserción o una supresión en el fichero a cada una de las rondas del programa.

Así que tendremos:

```
100 GO SUB iniciar
110 INPUT 'Insertar o Suprimir (a/b)';q$
120 IF q$='a' THEN  GO SUB cosecha: STOP
130 IF q$='b' THEN  GO SUB desecha: STOP
140 GO TO 110
```

SUPRESION

Vamos a por la rutina "desecha" (porque es la más fácil de las dos). Supongamos que la vamos a colocar a partir de la 6000. Necesitamos establecer la dimensión de una tabla literal que refleje el nombre del artista y la fecha de compra, con lo que identificaremos el disco unívocamente. Desde luego, eso supone que no has comprado dos discos del mismo artista en el mismo día. Esta posibilidad de ambigüedad, causa problemas a menudo al diseñar el fichero, y la manera usual de evitarlos es añadir un campo extra que se denomina **clave**, al comienzo de cada registro, y que contiene un número asignado únicamente a ese registro. Los números de cuenta en los bancos es un ejemplo de esto. En cualquier caso, suponiendo que todo está OK, necesitaremos:

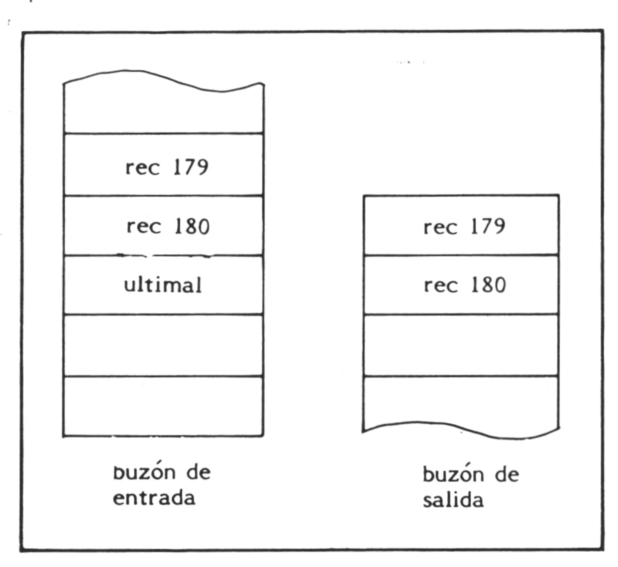
```
6000 DIM a$(336)
6010 INPUT 'Artista ';a$( TO 30)
6020 INPUT 'Fecha de compra ';a$(31 TO 36)
```

que indaga el disco que se quiere suprimir con el mismo formato en que escribimos la rutina **inrec**. Bien; todo lo que tenemos que hacer ahora es sacar los registros del fichero de entrada, ver si concuerda con este que queremos suprimir, y si no, volver a meterlos en el fichero de salida:

```
6030 GO SUB pinza
6040 IF r$( TO 36)=a$ THEN  GO TO 6030
6050 GO SUB punza
6060 GO TO 6030
```



Es simple, ¿verdad?. Desafortunadamente, es demasiado simple en el medio. Pero consideremos lo que sucede cuando los acercamos al extremo del fichero.



Supongamos que acabamos de sacar el registro 18Ø, y de ver que no es candidato a la supresión, por lo que ha sido transferido al buzón de salida mediante la rutina **punza**. Ahora, de nuevo citamos a **pinza** y al encontrarse con "ultimal", la rutina dice que ha alcanzado el final del fichero y el programa se detiene con un mensaje de error. Ahora puede que digas "eso no es ningún problema real, porque ya hemos finalizado de todas formas esta etapa".

Pero no lo hemos hecho del todo, porque en el buzón de salida todavía tenemos los registros 179 y 18Ø que no han sido punzados en cinta, y además, no hay marca de fin de fichero en el de salida, por lo que te puedo garantizar que sucederán misteriosos percances, si intentamos leer este fichero que acabamos de crear. A propósito, cuando ocurre este tipo de cosas, se dice que el buzón de salida no ha sido vaciado o descargado (en inglés se dice "flushed" porque su buzón lo asimila más a una cisterna de agua que descarga para limpiar un determinado depósito).

Hay unas cuantas maneras de salir de este atolladero en que hemos -mejor dicho hecaido. Quizá, la más simple sea tener dos marcas de fin de fichero, una para beneficio del control de ficheros (ultimal) y otra para beneficio del usuario (usaremos "}}").

Por lo tanto, necesitamos reescribir la rutina **cierre** para que genere ambas marcas de fin de fichero:

```
9740 LET r$=' }} ':
9750 GO SUB punza
9760 LET r$='ultimal'
9770 GO SUB punza
9780 RETURN
```

(Obviamente, tenemos que identificar **cierre** para beneficio de BASIC; podemos añadir en la línea 3:

3 LET cogebloque = 99ØØ: LET cierre = 974Ø

Podemos ahora modificar la rutina **desecha** para comprobar la marca de final de fichero del usuario, y cerrar el fichero de salida cuando se encuentre dicha marca.



```
6000 DIM a$(336)
6010 INPUT "Artista ";a$( TO 30)
6020 INPUT "Fecha de compra ";a$(31 TO 36)
6030 GO SUB pinza
6040 IF r$( TO 2)="}}" THEN  GO SUB cierre: RETURN
6050 IF r$( TO 36)=a$ THEN  GO TO 6030
6060 GO SUB punza
6070 GO TO 6030
```

Observa que en las líneas 6Ø4Ø y 6Ø5Ø, solamente se usan para comparaciones de los primeros dos y los primeros 36 octetos de r$, respectivamente. Es importante, y fácil de olvidar si no se es cuidadoso. Aquí el detalle, es que r$ tiene 336 octetos de largo, e incluso cuando está vacío, "}}" no es la misma cosa que "}} + 334 blancos" en cuanto a BASIC concierne.

INSERCION

Vamos ahora con la rutina **cosecha.** No hemos dicho nada todavía sobre el orden en que los registros van a quedar depositados en la cinta. Supongamos por un momento, que es orden alfabético según el nombre del artista; pero si hubiera más de un disco del mismo artista, el orden no estaría definido. Así que nuestro problema, en general, se puede enunciar así:

1. Imponer datos del nuevo disco
2. Leer un registro del fichero.
3. Si es final de fichero: pues cerrar fichero y regresar.
4. Si el nombre del artista en el registro del fichero, va delante del nombre del artista en el registro a añadir, se escribe el registro del fichero en el fichero de salida y se vuelve a efectuar la operación 2.
5. Se escribe el registro a añadir.
6. Escribe registro del fichero.
7. Vuelve a la operación 2.

En otras palabras, hacemos exactamente lo que haríamos con un fichero de fichas en cartulina: preparar la nueva ficha, repasar el fichero hasta que encontremos el lugar correcto para la nueva ficha, y meterla en él. La única diferencia es que en el caso del fichero mecanizado, estamos **materialmente** trasvasando los registros de un lugar a otro (entrada a salida) cada vez que los leemos. Es como si tuvieramos dos cajetines con fichas, y las reglas fueran que en cuanto leyéramos una del cajetín de entrada, tenía que ser copiada en otra ficha del cajetín de salida.

En nuestro algoritmo hay, sin embargo, una pifia muy sutil. Supongamos que el último registro de nuestra colección corresponde a Suzy Quatro, y queremos añadir un disco de Yango. El procedimiento definido arriba, funciona por todo el fichero, trasvasando registros a medida que pasa, hasta que llegue finalmente al disco de Suzy Quatro. En ese momento detecta la marca de final de fichero, así que cierra el fichero dejando los detalles del disco a añadir todavía en la memoria!. Por tanto necesitamos modificar la operación 3:

3. Si es final de fichero, pues comprueba, cierre de fichero, termina.

Siendo **comprueba** una rutina que indague si el registro a añadir ya ha sido transferido al fichero, y si no lo ha sido, ella misma lo transferirá.



Vamos a redactar la rutina **cosecha** a partir de 5ØØØ. Primero tenemos que imponer los detalles del disco a incorporar a la colección. Pero como ya hemos preparado una rutina que acepta la serie completa de detalles del disco: **inreg,** simplemente la citaremos. No hay necesidad de volver a inventar la rueda...

```
5000 GO SUB inreg: LET trasvase=0
5010 GO SUB pinza
5020 IF r$( TO 2)='}}' THEN
     GO SUB comprueba: GO SUB cierre: RETURN
5030 IF r$( TO 30)<a$( TO 30) THEN
     GO SUB punza: GO TO 5010
```

Antes de que avancemos demasiado, hay un par de cosas que necesitan breves explicaciones. Primeramente, la instrucción que hace que el trasvase sea igual a cero. La rutina **comprueba** va a necesitar saber de alguna manera, si los detalles del disco adicional han sido transferidos al fichero de salida o no. En cuanto se imponen por teclado los detalles, hacemos que el trasvase sea igual a cero. Cuando la ficha pertinente ha sido trasvasada al fichero de salida, tendremos que poner trasvase a uno. Por lo tanto, **comprueba** sólo tiene que mirar trasvase para ver si es cero. Si lo es, todavía hay que sacar al fichero de salida la ficha añadida.

En segundo lugar, la línea 5Ø3Ø compara dos variables litéricas: r$ con el valor entregado por la rutina **pinza,** y a$ que es el devuelto por la rutina **inreg.** (r$ en un principio también fue impuesta mediante **inreg,** pero inmediatamente después fue machacado por **pinza**). El uso del símbolo "menor que" tiene aquí el significado de "va delante alfabéticamente", de manera que tratar con valores litéricos en orden alfabético no presenta ningún problema.

Para continuar:

```
5040 LET b$=r$
5050 LET r$=a$
5060 GO SUB punza
5070 LET trasvase=1
5080 LET r$=b$
5090 GO SUB punza
5100 GO TO 5010
```

[deposita el último registro sacado del fichero]

[transpasa el incorporado a r$ para escribirlo...
y lo escribe]

[indica que ya ha sido escrito]

[recupera el último registro en r$ para escribirlo...
y lo escribe]

Ahora, finalmente, para escribir **comprueba** (a partir de 52ØØ):

```
5200 IF trasvase =
     1 THEN  RETURN
5210 LET r$=a$
5220 GO SUB punza
5230 RETURN
```

[no se necesita ninguna acción porque ya se ha sacado al fichero de salida el nuevo registro]

[deposita en r$ el registro a añadir
y lo escribe]





Proyecto

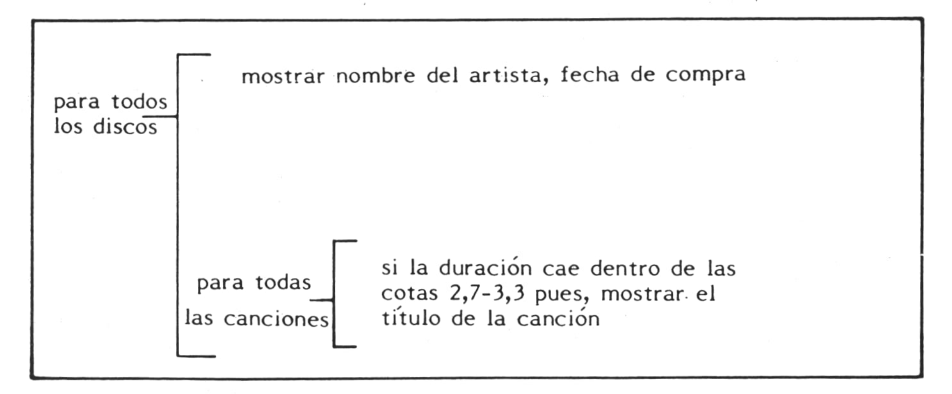
Todo lo precedente supone que solamente va a hacerse una alteración al fichero por pasada. Pero si sales y te vuelves loco comprando discos y consigues 15 nuevos discos, y luego regalas 4 de tus viejos a Pepito, tendrás que pasar el programa de mantenimiento de ficheros 19 veces para mantener actualizado el fichero.

Intenta escribir un programa de mantenimiento que al principio le impongas por teclado todas las variaciones. Las deposite temporalmente en una tabla las inserciones, y en otra las "delecciones". Luego tendrás que cotejar cada registro del fichero con cada uno de los elementos de las dos tablas para decidir si tienes que suprimirlo, trasvasarlo al fichero de salida, o efectuar previamente una inserción. Observa que cada uno de los discos a incorporar necesita su propio testigo de trasvase (i.e. por eso tendrá que haber una tabla de trasvases y no una variable simple). Observa además, que el orden en que reflejes los añadidos no es importante, porque el programa buscará en todo momento para ver si tiene que insertar algo. No te olvides de admitir el hecho que puede ser necesario hacer más de una inserción entre dos registros dados del fichero. Es un pensamiento interesante el realizar la creación del fichero simplemente con un par de registros, y luego usar el propio programa de mantenimiento para añadir el resto de los registros: ¡el fichero se generará automáticamente en orden alfabético!

ESCRUTANDO EL FICHERO

Habiendo gastado un tiempo considerable en crear este fichero, debieramos encontrar algún uso para él. Supongamos que queremos preparar una cinta con música de fondo para nuestro guateque. Queremos una cierta gama de canciones, todas de 3 minutos aproximadamente, por lo que nos gustaría es una lista de las canciones que duran entre 2,7 y 3,3 minutos, por ejemplo.

Así que queremos un programa que haga:



Es bastante simple:

```
100 GO SUB iniciar
110 GO SUB pinza
120 IF r$( TO 2)='}}' THEN  STOP
130 PRINT r$( TO 30)
140 PRINT r$(31 TO 36)
```



```
150 FOR t=1 TO 12
160 LET comienzo=57+(t-1)*25
170 LET d=VAL r$(comienzo TO comienzo+4)
180 IF d<2.7 OR d>3.3 THEN  GO TO 200
190 PRINT r$(comienzo-20 TO comienzo-1)
200 NEXT t
210 GO TO 110
```

Observa que, dado que sólo **leemos** el fichero, no hay necesidad de hacer operaciones de cierre, cuando se detecta su final en la instrucción 12∅.

La única argucia aquí, es evaluar la duración de cada canción como un número. Primero tenemos que detectar la parte significativa de r$ (línea 16∅). Luego es necesario crear un valor numérico a partir de esta variable litérica (línea 17∅) de forma que podamos comparar números como hacemos en la línea 18∅. Evidentemente, los valores 2,7 y 3,3 pudieran simplemente asignarse a variables al principio del programa. Así se nos permitiría hacer otras preguntas, como:

> "Enumera todas las canciones con una duración mínima de 4,5 minutos" (con eso nos bastaría dar como valor de la cota inferior 4,5;y de la superior algo ilógico pero aceptable como 9999).

O bien:

> "Lista todas las canciones que duren exactamente 3 minutos" (si hiciéramos ambas cosas igual a 3).

Proyecto

Detectar otros rasgos específicos del fichero es igual de fácil. Ensaya los siguientes:

1. Enumerar todas las canciones de un artista dado.
2. Lo mismo que para 1 pero con la fecha de compra incluida entre dos fechas dadas. (Ligeramente más difícil de lo que parece, a causa de las comparaciones de fechas).
3. Relaciona todas las canciones en cuyo título aparezca la palabra "rock" o cualquier otra palabra impuesta por teclado.

REGISTROS DE LONGITUD VARIABLE

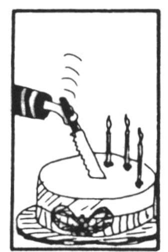
He dicho al comenzar que ibamos a suponer que todos los discos tenían 12 canciones, y cuyo título ocupaba exactamente 2∅ octetos, y otras cosas. Lo que he creado es un fichero cuyos registros están garantizados que son todos de 336 octetos de longitud. Dudo que te sorprenda saber que de tales ficheros decimos que tienen registros de longitud prefijada o constante. Además, dentro de cada registro, los campos también son de longitud prefijada, de manera que cada título de canción tiene que ser completado con espacios (o abreviado cuando es largo) para que tenga 2∅ octetos. Ahora bien, en un mundo ideal, sería hermoso permitir campos con longitud variable (terminando cada uno con un delimitador de campos de alguna clase) y registros de longitud variable (también terminados con algún otro delimitador). De esa manera, no desperdiciaríamos 27 octetos con cada album de "OLA", y no habría 8 títulos nulos para cada una de las sinfonías de música clásica (suponiendo que tratamos cada uno de sus movimientos como una canción).

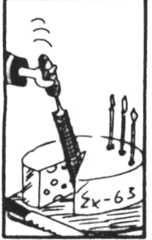


Sin embargo, considera que a veces, lo que se gana en las habitaciones se pierde en los pasillos. Primeramente, el programa que segrega los campos de un registro se hace más complicado. Tenemos que escrutar octeto a octeto, para detectar los delimitadores de campos, y no podremos hablar sobre el "quinto campo" porque puede que no haya. En segundo lugar, los utensilios de programación -las rutinas- para leer, escribir y tratar los bloques en que se segmenta el fichero, se vuelven bastante peliagudas. Después de todo, con nuestros campos y registros prefijados, fue fácil definir un bloque (y también eficaz) simplemente como una lista, con tantos elementos como registros por bloque, y de tamaño igual al número de octetos por registro. Si cambia el número de octetos por registro, según el registro que sea, ya no podemos pensar de forma tan cómoda. Por otro lado, si mantenemos a un tamaño constante los buzones, cambia entonces el número de registros por bloque en función del tamaño de los registros concernientes!

No quiero dar la impresión de que tratar con registros de longitud no constante, está fuera de la pericia de un hombre normal; sólo que puede que no sea tan sensato como a primera vista aparece. La cuestión, es que si aumentamos la complejidad de nuestras rutinas, también incrementamos su tamaño, lo que tiene como consecuencia la disminución de la memoria que queda libre para el programa de usuario y para los buzones de los ficheros. Además, y a no ser que se esté desperdiciando un montón de espacio al usar los formatos de longitud prefijada, la introducción de los delimitadores necesarios puede usar la mayoría del espacio que pretendemos ahorrar. A menudo, una elección cuidadosa de las longitudes de los campos al usar un sistema con tamaños prefijados, constituye una respuesta perfectamente adecuada. Debiéramos también considerar cuidadosamente el contenido de los registros: la información. ¿**Realmente** queremos discos de música clásica y música popular en el mismo fichero? ¿No sería mejor tener dos ficheros, de manera que pueda definirse cada uno de una forma sensata, sin tener que preocuparse de las peculiaridades de uno y de otro?

¿Eres capaz de deducir del chiste, cómo se dice en inglés tanto limas como ficheros?

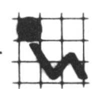






En cualquier caso, nunca debiéramos olvidar que estamos trabajando con cintas magnéticas como depósito de respaldo para nuestro fichero, lo que es barato, en lugar de con memorias que no son tan baratas. De hecho, 5 Kiloctetos (Koc) de información, te costarán alrededor de 2 pesetas en cinta magnética. Es fácil de calcular -recuerda simplemente que a un ritmo de transferencia de alrededor de 2ØØ octetos/segundo, se tardarán 25 segundos en depositar 5 Kocs en la cinta, y luego haciendo 25 ÷ (60 x 60) de lo que hayas pagado por tu última cassette C60. Probablemente te saldrá que mis cifras son pesimistas. 5 Kocs en memoria de semiconductores, es, incluso hoy, en que los precios de las memorias están bajando constantemente, todavía te costarán alrededor de 2.500 pesetas. No creo que haya duda.

Si lo mencionado anteriormente te suena como excusas plausibles de alguien que no quiere tener que trabajar duramente a menos que realmente le fuercen a ello, habrías intuido perfectamente mi talante al escribirlo. Los ordenadores se supone que están para hacer la vida más fácil, no más difícil.



Una propuesta modesta

Hay una particularidad molesta en el sistema de ficheros en cassette, tal y como lo hemos dejado: **iniciar** pregunta al usuario los tamaños del registro y de bloques, incluso para ficheros que existen ya.

Modifica el programa de manera que se escriba un bloque "cabecera" al principio de cualquier fichero de salida, en que estén reflejados los detalles de tamaño de los registros y bloques de ese fichero. Luego, si se contesta a **iniciar** con un nombre de fichero de entrada, esa rutina ya no preguntará los tamaños de los registros y los buzones, sino que los leerá a partir del bloque cabecera de ese fichero de entrada. Es lo que en ordenadores mayores, decimos "ficheros con etiquetas" **("label").**

(Para una propuesta **menos** modesta, véase el capítulo 17).



Hay mentiras, condenadas mentiras,
y... listados de ordenador.

10 La estadística de forma simple

Servidores de la Salud Nacional, S.A. (SSA) emplea 24 personas a 2.000 pesetas por semana. El Director General consigue 5 millones al año. Cuando el Sindicato Regional de Empleados Servidores de la Salud (SRESS) fueron a la huelga pidiendo más paga, la dirección sacó una página completa en Diario 16 indicando al público que el salario **medio** era de 11.840 pesetas por semana. En la entrevista, el director dijo textualmente: "Estos señores deben ir a trabajar inmediatamente y dejar de quejarse: 11.840 pesetas es un salario semanal bastante justo".

Las estadísticas pueden ser usadas, y mal usadas. Las **medias** pueden ser una medida correcta de lo normal, el valor típico, pero algunas veces. Pueden también estar distorsionadas por la excepción singular que está completamente fuera de línea, como en este caso. El **cálculo** es correcto:

$$\text{Media} = \frac{\text{Nómina total semanal}}{\text{Número de empleados}} = \frac{24 \times 2.000 + 200.000}{25} = 9.920$$

Pero la **interpretación** -"la mayoría de los empleados sacan unas 9.920 ptas."- obviamente no lo es.

Esto intenta mostrar que la comprensión de las estadísticas básicas es digna de adquirirse. En este caso, SRESS publicó su propio anuncio, indicando que el indicador estadístico más apropiado no es la **media,** sino la **moda**: el salario más común, que en este caso es 2.000 ptas. Reconociendo la lógica impecable de este argumento, SSA botó a su director general, dando el control a un comité de empleados y subió sus salarios semanales, excepto para el presidente del comité que pasó a 18.000 ptas. semanales y que era el que sabía calcular medias y modas.

Desde luego, no siempre sucede de esa manera, sino que más bien es el que podría ser presidente del comité el que sale botado de la empresa...

PRESENTACION DE DATOS: HISTOGRAMAS

No es éste el lugar para enseñarte teoría estadística. Lo que voy a hacer es escribir algunos programas que te dejan explorar ideas estadísticas sin tener que ir profundamente a las entrañas de las mismas. De esta manera, puedes empezar a percatarte de lo que significan en la práctica.

Y una parte importante de estadística concierne a la manera en que se presentan los datos. Siendo el Spectrum una bestia visual, me concentraré en dos formas standard de representación gráfica: el **histograma** y los **quesos (sectograma).**

Un histograma muestra cuán a menudo aparece un determinado "valor". Por ejemplo, supongamos que lanzó un dado veinticuatro veces, y que los resultados obtenidos han sido:

el 1 lo he sacado 3 veces
el 2 lo he sacado 6 veces
el 3 lo he sacado 5 veces
el 4 lo he sacado 5 veces
el 5 lo he sacado 4 veces
el 6 lo he sacado 5 veces



Por tanto, el histograma tendrá seis barras verticales, numeradas de 1 a 6, con las alturas correspondientes. Así la barra 1 tendrá de altura 3, la barra 2 de altura 6, y demás; Figura 10.1.

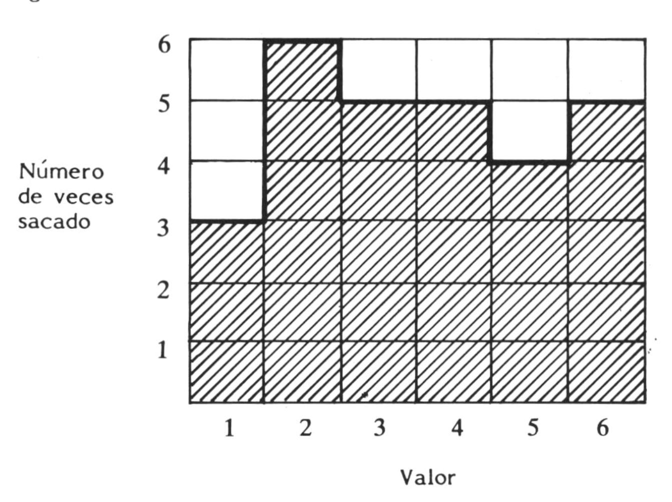


Figura 10.1 El histograma muestra el número de veces en que apareció un determinado valor al lanzar el dado.

El valor promedio, o **media** obtenido en el lanzamiento, viene dado por

$$\frac{1 \times 3 + 2 \times 6 + 3 \times 5 + 4 \times 5 + 5 \times 4 + 6 \times 5}{28} = 3{,}57$$

Observa cómo cada puntuación se multiplica por el tamaño de la barra. Si el dado no está trucado, a la larga, aparecerían números cada vez más iguales para cada uno de los valores de 1 a 6, con un valor medio que sería:

$$\frac{1 + 2 + 3 + 4 + 5 + 6}{6} = 3{,}5$$

En esta estadística particular, la práctica concuerda muy bien con la teoría. Observa, sin embargo, que el valor más común en el experimento -la moda- ha sido el número 2 que apareció 6 veces.

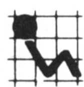
La moda te dice dónde está el punto más alto del histograma, la media te dice dónde debes trazar una línea vertical, de manera que se equilibren las barras que pasan por encima con las barras que quedan por debajo. Las barras no necesitan tener la misma altura; y la media sólo da una idea del valor "típico" si los números no están muy esparcidos. En este caso, no están **muy** esparcidos o **dispersos.**

Hablaremos sobre cómo medir el grado de dispersión posteriormente; por el momento, todo lo que quiero realmente es la idea de histogramas. Para acostumbrarte a ella, aquí hay un programa que presenta un histograma de lanzamientos de un dado, cuyos resultados vas imponiendo después de lanzar. Te dice, por tanto, cuál es el valor medio en cada una de las sucesivas etapas.



```
  10 LET ini=500
  20 LET valin=1000
  30 LET hist=1500
  40 LET media=2000
  50 LET wtot=0: LET num=0
 100 DIM d(6)
 200 GO SUB ini
 210 GO SUB valin
 220 GO SUB hist
 230 GO SUB media
 240 GO TO 210
 500 REM ini
 510 PLOT 103,175: DRAW 0,-160: DRAW 144,0
 520 PRINT AT 21,13;' 1  2  3  4  5  6 '
 530 FOR i=1 TO 6
 540 PLOT 104+24*i,15: DRAW 0,-5
 550 NEXT i
 560 FOR i=0 TO 20
 570 PLOT 103,15+8*i: DRAW -5-5*(i=10 OR i=20),0
 580 NEXT i
 590 PRINT AT 0,9;'20';AT 10,9;'10'
 600 RETURN
1000 REM valin
1010 IF INKEY$<>'' THEN  GO TO 1010
1020 IF INKEY$='' THEN  GO TO 1020
1030 LET c=CODE INKEY$-48
1040 IF c<0 OR c>6 THEN  GO TO valin
1050 IF c=0 THEN  STOP
1060 LET d(c)=d(c)+1
1070 RETURN
1500 REM hist
1510 IF d(c)=21 THEN  INPUT 'No me cabe histograma';x$: STOP
1520 PRINT AT 20-d(c),3*c+10; INK c;' □ '
1530 RETURN
2000 REM media
2010 LET num=num+1: LET wtot=wtot+c
2020 LET a=.01*INT (100*wtot/num)
2030 PRINT AT 2,1;'Media='
2040 PRINT AT 4,2;a;'□□□'
2050 RETURN
```

Tecléalo y a rular. Saca un dado y lánzalo: pulsa el número que resulte (es decir, si sale un 4, pulsa la tecla 4. No necesitas teclear -pulsar ENTER después del dato-). Se va construyendo un histograma coloreado. Para cada número del 1 al 6, presenta cuántas veces has obtenido dicho número. También se imprime la media a medida que lanzas el dado. Si cualquiera de los números sale más de 20 veces, el programa se detiene con un mensaje (realmente tienes que presionar una tecla para conseguir que se detenga). Para terminar antes de eso, pulsa 0.



Verás sin dificultad cómo funciona este programa. La rutina principal está en las líneas 2ØØ-24Ø. En **ini** se establecen los ejes y la escala; **valin** coge el valor pulsado; **hist** dibuja el diagrama; **media** la calcula y la imprime.

El asunto de la parte entera (INT) en la línea 2Ø2Ø, es simplemente una manera de asegurar que sólo se imprimen dos cifras fraccionarias (o menos). Es un truco aprovechable (para tres cifras decimales, usa LET a =.ØØ1 * INT (1ØØØ * wtot/num) y así sucesivamente, con un cero extra en ambas constantes para cada cifra decimal extra que desees.

REPRESENTACION DE DATOS: SECTOGRAMAS

Llamadas vulgar y adecuadamente "quesos", muestran cómo se divide el pastel entre los diferentes comensales. Se corta un círculo con sectores de área proporcional a los resultados. Ya sabes de qué se trata.

El siguiente programa es un "cortador de quesos" automático. Recibe como entrada una serie de artículos con nombre (digamos conceptos de gasto) y produce un diagrama de sectores. Funciona mejor con 1Ø o menos partidas, y a poder ser, ninguna partida debiera ser de menos del 3% del total. También **funciona** si no se cumplen estos criterios; pero por sí mismos, los quesos no serían aprovechables porque tendrían demasiadas porciones o porciones que apenas puedes ver.

```
200 REM data in
210 INPUT 'Numero de porciones?';n
220 DIM i$(n,9): DIM v(n)
230 DIM a(n+1)
240 FOR i=1 TO n
250 INPUT 'Nombre de porcion?';i$(i)
260 INPUT 'Valor de porcion?';v(i)
270 PRINT AT i,5;i$(i),v(i);'□□□'
280 INPUT 'Es correcto? s/n';q$
290 IF q$='n' THEN  GO TO 250
300 LPRINT i$(i),v(i): REM solo si tienes impresora
310 NEXT i
320 LET tot=0
330 FOR i=1 TO n
340 LET tot=tot+v(i)
350 NEXT i
500 REM queso en porciones
510 CLS : CIRCLE 84,84,75
520 LET ang=0
530 FOR i=1 TO n
540 LET ang=ang+v(i)*2*PI/tot
550 LET a(i+1)=ang
560 PLOT 84,84
570 DRAW 75*COS ang,75*SIN ang
600 REM rotulos de porciones
610 LET u=.5*(a(i)+a(i+1))
620 PRINT AT 11-7*SIN u,10+7*COS u;i
630 NEXT i
700 REM rimgla
710 FOR i=1 TO n
720 PRINT AT i,21;i;':';i$(i)
730 NEXT i
740 RANDOMIZE USR 65044
```




Las líneas 3ØØ y 74Ø debieran dejarse fuera a no ser que tuvieras impresora y realmente quisieras estampar los resultados. Las líneas 28Ø y 29Ø proveen la manera de corregir errores si los cazas inmediatamente: si esta posibilidad te molesta, suprímela. (A propósito, no necesitas pulsar "s" para la contestación sí": cualquier tecla excepto "n" funcionará. Obviamente, la mejor es ADENTRO).

Por ejemplo, teclea las siguientes cifras, que dan el producto nacional bruto (por cabeza) de los países del Mercado Común en 1978 (los nombres de países en inglés, por supuesto, para que vayas preparándote para el futuro).

Número de porciones: 9 Nombre de la porción	Valor de la porción	s/n
Belgium	129	n para corregir
Denmark	144	"
France	116	"
Germany	137	"
Holland	123	"
Ireland	5Ø	"
Italy	6Ø	"
Luxembourg	126	"
UK	73	"

La Figura 10.2 muestra el diagrama de quesos resultante. Experimenta usando otras cifras, reales o imaginarias.

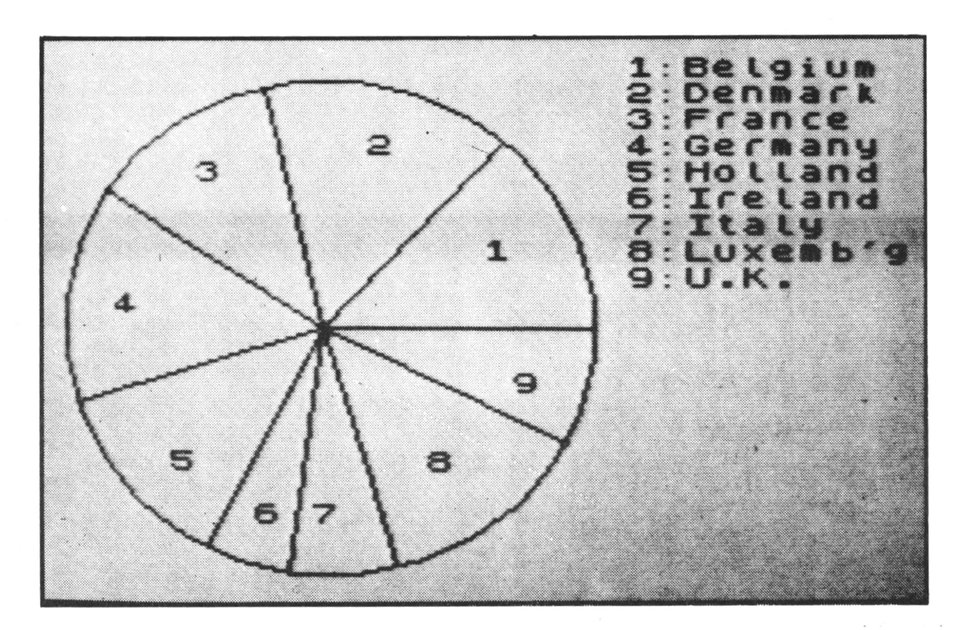


Figura 10.2 Cómo se dividen el pastel en el Mercado Común...



MEDIAS, MODAS Y TODO LO DEMAS

Ya he explicado medias y modas (hay otra criatura que se llama mediana, pero no te confundas). La media es el valor "promedio", la moda (y puede haber más de una) es el valor "más común", o sea, el que más veces sale. También he mencionado que la media es "típica", razonablemente, siempre que los datos no estén muy dispersos.

Para medir la dispersión, los estadísticos inventaron un chisme llamado **desviación standard.** Es la raiz cuadrática media de las desviaciones sobre la media, si entiendes lo que he dicho.

Tienen también una curva favorita, que llaman curva **normal** que se usa para suavizar los cantos de los histogramas: se elige de forma que tenga la misma media y la misma desviación standard, pero tiene la forma de joroba de camello.

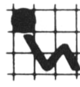
En lugar de presentarte un montón de matemáticas (lo cual puedes estudiar en un texto de estadística y casi cualquiera valdrá) he escrito un programa que te permite producir tus propios histogramas, y luego te dice la media, la moda, la desviación standard, y como premio, también te pinta la curva normal correspondiente a ese histograma.

Si tú (o tu hijo) está estudiando estadística en la escuela, este programa le ayudará (a él o a ella) para lograr percatarse de lo que representan estas cosas.

```
  10 LET ini=500
  20 LET dibuja=1000
  30 LET estadistica=2000
  40 LET normal=3000
  50 DIM a(24)
  60 LET paso=3
 100 GO SUB ini
 110 GO SUB dibuja
 120 GO SUB estadistica
 130 GO SUB normal
 140 STOP
 500 REM ini
 510 CLS
 520 PLOT 15,165
 530 DRAW 0,-150
 540 DRAW 240,0
 550 FOR t=1 TO 24
 560 PLOT 15+10*t,15: DRAW 0,-3,-3*(t=10 OR t=20)
 570 NEXT t
 580 FOR t=1 TO 15
 590 PLOT 15,15+10*t: DRAW -3-3*(t=10),0
 600 NEXT t
 610 PRINT  OVER 1;AT 21,0;"  [13 blancos]   10  [11 blancos]  20"
 620 PRINT AT 7,0;"1"
 630 PRINT AT 8,0;"0"
 640 RETURN
```




```
1000 REM dibuja
1005 LET h=15
1010 FOR i=1 TO 24
1020 IF INKEY$<>'' THEN  GO TO 1020
1030 IF INKEY$='' THEN  GO TO 1030
1040 LET k$=INKEY$
1045 LET h=h+10*(CODE k$-53)
1050 IF h<15 THEN  LET h=15
1055 IF h>165 THEN  LET h=165
1060 LET a(i)=(h-15)/10
1070 PLOT 15+10*i-10,15: DRAW 0,h-15: DRAW 10,0: DRAW 0,15-h
1080 NEXT i
1090 RETURN
2000 REM estadistica
2005 LET tot=0: LET norm=0: LET m=0: LET mv=0
2010 FOR i=1 TO 24
2015 IF a(i)>mv THEN  LET mv=a(i): LET m=i
2020 LET tot=tot+i*a(i)
2025 LET norm=norm+a(i)
2030 NEXT i
2040 LET av=tot/norm
2050 PRINT AT 0,3;'Media= ';av
2055 PRINT AT 1,3;'Moda= ';m
2060 LET std=0
2070 FOR i=1 TO 24
2080 LET std=std+a(i)*(i-av)*(i-av)
2090 NEXT i
2100 LET std=SQR (std/norm)
2105 LET std=.01*INT (100*std)
2110 PRINT AT 2,3;'Desviacion Estandar= ';std
2120 RETURN
3000 REM normal
3010 FOR i=0 TO 240  STEP paso
3020 LET y=EXP (-(.5+i/10-av)*(.5+i/10-av)/(2*std*std))/
     (std*SQR (2*PI))
3030 LET y=y*tot
3035 IF y>=150 THEN  GO TO 3060
3040 IF i=0 THEN  PLOT i+15,y+15
3050 IF i>0 THEN  DRAW paso,y+15-PEEK 23678
3060 NEXT i
3070 RETURN
```



USANDO EL PROGRAMA

El programa está diseñado para que sea fácil establecer los histogramas de comprobación; y como consecuencia, los datos se imponen de forma amistosa y poco ortodoxa y hasta puede enfurecerte hasta que te acostumbres. Funciona de esta manera: las barras en las posiciones1-24 se teclean sucesivamente. Inicialmente, la altura es Ø (y siempre será entre Ø y 16). La **siguiente** altura de barra es 5 - k veces **mayor** que el número de la tecla k de la fila superior que se haya pulsado. Es decir, las teclas numéricas controlan el **cambio en altura** de una barra desde una columna a la siguiente, como presentamos a continuación.

Tecla	Efecto sobre la altura de la barra
1	4 más pequeña
2	3 más pequeña
3	2 más pequeña
4	1 más pequeña
5	igual
6	1 más alta
7	2 más alta
8	3 más alta
9	4 más alta

Por ejemplo, rúlalo y luego pulsa (no teclees) las teclas

5 5 6 6 6 7 8 5 2 3 4 4 4 5 5 5 5 5 5 5 5 5 5 5

para obtener el resultado de la figura 10.3.

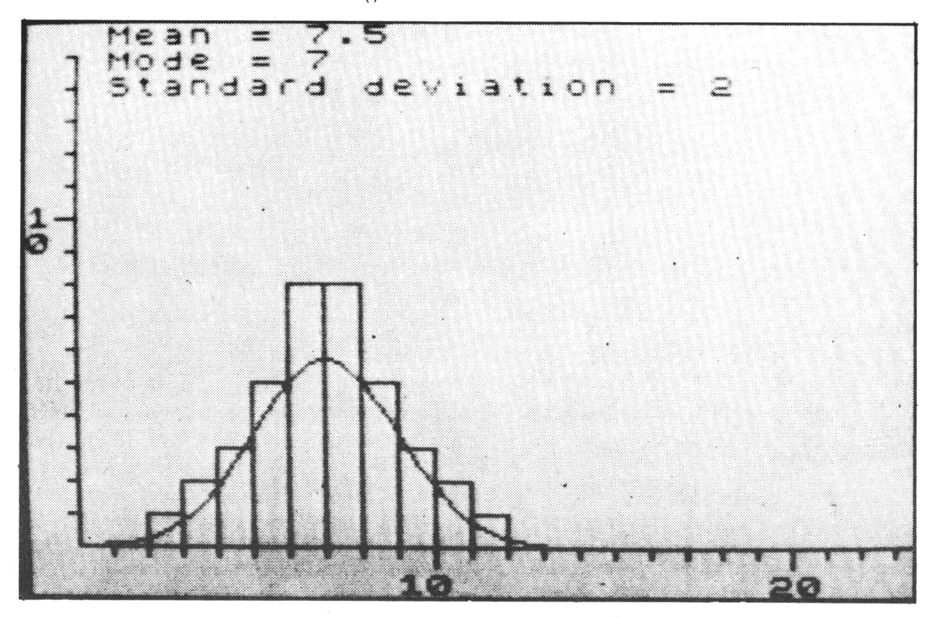


Figura 10.3 Curva normal que corresponde a un histograma con bastante precisión...

Inventa tus propios histogramas, y comprueba que:

1. La media es un valor "promedio" razonable. Si cortaras el histograma una vez forjado en una lámina metálica, pesaría igual que un rectángulo con altura igual a la media.

2. La moda es el primer valor de "pico". (Los otros lugares que también tengan esa altura, son también modas. Pero el programa no observa nada más que el primer pico; sería fácil cambiar eso).
3. Los histogramas que están más "dispersos" como
   5 5 6 6 6 7 8 5 5 5 5 5 2 3 4 4 4 5 5 5 5 5 5
   tienen mayor desviación standard; los histogramas que están más "amontonados"
   5 5 5 5 5 5 9 9 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5
   tienen desviación standard menor.
4. La curva normal es una aproximación razonable a aquellos histogramas que tengan sólo un pico, no estén demasiado dispersos y sean bastante simétricos...

Pero puede producir confusión total si se aplica a otros histogramas, por ejemplo, para el que tiene dos jorobas como

5 5 5 7 7 7 9 1 1 4 4 5 5 5 8 8 3 3 3 5 5 5 5 5

que da el resultado de la figura 10.4.

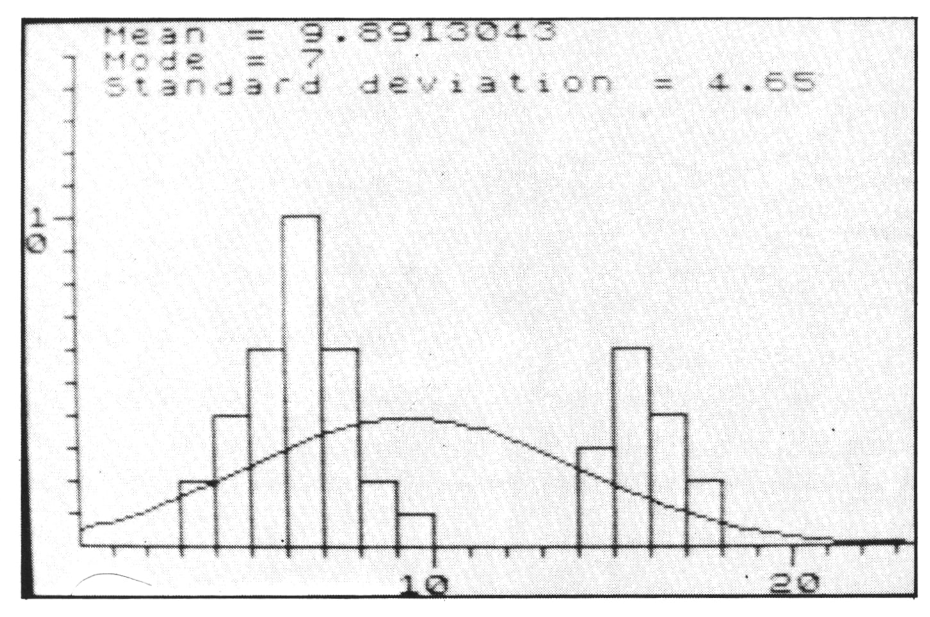


Figura 10.4 ...y bastante poco adecuada.

Proyecto

Cambia las líneas 1Ø45-1Ø6Ø de manera que lo que teclees sea **directamente** las alturas de las barras. (Es fácil de hacer: la única razón de que usaras la otra manera en el programa es que produce los histogramas muy rápidamente, y sin tener que pensar mucho, una vez que te acostumbras a ellos, de manera que es ideal para los experimentos.

Experimento

Lanza 4 dados, y anota el total. Hazlo 1ØØ veces. El histograma que resulte, dálo como dato de entrada del programa anterior (por medio del proyecto) y observa cuáles son los resultados. ¿La curva normas se adapta estrechamente o no?

Simulación

Usa los números aleatorios del Spectrum para simular el lanzamiento de 4 dados: repite el análisis estadístico.



Un poco de atención a los detalles puede hacer maravillas -tales como traducir la Spectrumática en álgebra normal...

11 Mejorando la imagen

Muchos programas pueden resultar más atractivos, estableciendo caracteres definibles por el usuario para conseguir una representación más precisa del efecto buscado. Vamos a plantear un proyecto según esas miras, que acepte **polinomios** de varias variables x, y, z, en la forma que el Spectrum acostumbra, y hace que parezca algebra vulgar. (Si no te gusta el algebra, sigue conmigo, lo importante sigue siendo la computación).

En el álgebra ordinaria, los polinomios se escriben así:

$ax^2 + bx + c$

$2x^3 + 5y^7$

$a^3 + b^3 + c^3 - 3abc$

Pero el Spectrum usa "*" para la multiplicación, en lugar de escribir los símbolos uno a continuación de otro; y también usa "↑" para la exponenciación, en lugar de utilizar superíndices; con lo que resulta:

```
a * x↑2 + b * x + c
2 * x↑3 + 5 * y↑7
a↑3 + b↑3 + c↑3 - 3 * a * b * c
```

El primer paso es desarrollar caracteres para los exponentes 1, 2, ..., 9, ∅. La figura 11.1 muestra uno de los posibles trazados. Tecléalos como caracteres gráficos de la "a" a la "j", en la manera usual (Fácil Programación, pág. 49).





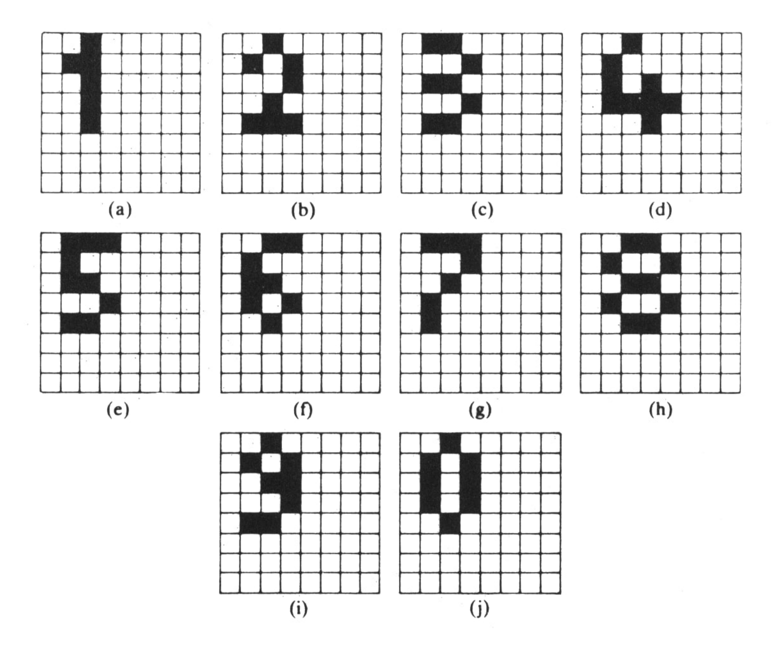
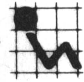


Figura 11.1 Trazado gráfico de exponentes.

Luego teclea el siguiente programa: Impón una expresión algebráica según Spectrumática y pasará a ejecutar tres tareas:

1. Quitará todos los asteriscos.
2. Convertirá todos los números que vengan después de ↑, en los exponentes definidos.
3. Quitará todas las ↑ una vez que haya efectuado la tarea 2.

```
  5 LET poten=200
  6 LET produ=600
  7 LET mostra=1000
 10 INPUT 'Expresion a 'MAQUILLAR'',,,e$
 20 LET l=LEN e$
 30 LET f$=''
 40 LET i=1
 50 IF e$(i)='?' THEN  GO TO poten
 60 IF e$(i)='*' THEN  GO TO produ
 70 LET f$=f$+e$(i)
 80 LET i=i+1
 90 IF i>l THEN  GO TO mostra
100 GO TO 50
200 REM potencias
210 LET j=i+1
212 IF j>l THEN  GO TO mostra
215 LET c=CODE e$(j)
220 IF c<48 OR c>57 THEN  GO TO 500
```



```
 230 LET f$=f$+CHR$ (c+95+10*(c=48))
 240 LET j=j+1
 245 IF j>1 THEN  GO TO mostra
 250 GO TO 212
 500 LET i=j
 510 GO TO 50
 600 REM productos
 610 LET i=i+1
 620 GO TO 50
1000 REM presenta resultado
1010 PRINT 'Basic    :□';e$'''Algebra :□';f$
```

COMPROBANDOLO Y PONIENDOLO A SALVO

Ensaya con las expresiones enumeradas anteriormente, en forma Spectrumática, que es el dato de entrada e$; comprueba que aparece el resultado correcto. Ahora ensaya algunas otras expresiones algebráicas como:

2 * x * y * z - 17 * a↑552

m↑77 + n↑88

a * b * c * d * e * f * g + 45 * h↑999 + 32

y las que quieras.

El programa no traga con **todo:** observa lo que hace con 2 * 2, por ejemplo! Pero ilustra la idea que queremos.

Para guardar este programa en cinta de forma aprovechable, debemos hacer un poco más de trabajo, porque los gráficos definidos por el usuario no quedan guardados automáticamente. Primero, debemos determinar dónde se encuentran.

En el Spectrum de 16K, comienzan a partir de la dirección 326ØØ. Pero puede que lo hayas cambiado (por medio de la variable sistemal UDG, que reside en la dirección 23675-6). De manera que puede que prefieras determinar la dirección precisa mediante

PEEK 23675 + 256 * PEEK 23676

Sin embargo, hay una manera más fácil, porque el carácter gráfico correspondiente a "a" tiene que comenzar en el área UDG. Por lo tanto

USR "a"

funcionará. (Expón en pantalla ambos, y lo verás).

Rebobina tu cinta hasta el lugar en que quieras comenzar a grabar, y añades una instrucción más como preparación para lo que viene a continuación, a saber

```
1 LOAD 'exp'CODE USR 'a',80
```

Luego guarda en la cinta todo el programa, usando

SAVE "algebra" LINE 1

con lo que tendrá un arranque automático comenzando en la línea 1.



Todavía no has acabado: tienes que usar ahora la grabación de áreas de octetos para que se conserven sus gráficas:

```
SAVE "exp" CODE USR "a", 80
```

Con esto, su ordenador recoge los 80 octetos del área de gráficos definibles por usuario, se comienza a partir de USR "a", y los deposita en la cinta. Hemos dicho 80 porque cada carácter ocupa 8 octetos, y tenemos 10 caracteres, de manera que el total es 8 * 10 = 80. Ahora ya sabes que la línea 1 añadida, invierte este procedimiento para recuperar esos octetos de la cinta y trasvasarlos a la memoria.

Si ambos han sido guardados en cinta, rebobina hasta la posición inicial, y teclea

```
LOAD "algebra"
```

Te saldrán los bufidos y pitidos habituales y las bandas rojas/azules/amarillas, y todo lo demás y el mensaje

```
Program: algebra
```

Deja la cinta marchando. "Algebra" automáticamente empezará por la instrucción de la línea 1; lo que provoca automáticamente que se recuperen de la cinta los gráficos definidos por el usuario y se carguen en memoria. Eso producirá más bufidos y pitidos y bandas, y el mensaje

```
Bytes: exp
```

después de lo cual, "algebra" pasará a la siguiente línea, y continuará funcionando como ya sabíamos. Detén la cinta y ya puedes pasar adelante.

Este es un ejemplo de cómo **encadenar** programas depositados en cinta, aprovechándose de que los comandos LOAD pueden usarse como instrucciones en un programa. Para ver otro ejemplo, pasa al capítulo 15; Cambiando el Repertorio de Caracteres.

Proyectos

1. Modifica "algebra" para que desarrolle productos de números, tales como 23 * 45, en lugar de pegarlos sacando 2345 (lo que es erróneo) tal y como hace la presente versión.
2. Modifícalo para que expresiones como x * x se conviertan en x^2; o x * x * x y x * x^2 en x^3, y demás.
3. Modifícalo para que **ordene** las variables alfabéticamente, de forma que abcbab se convierta en aabbbc o (mejor) a^2b^3c.
4. Modifícalo para quitar cualquier término del polinomio que resulte multiplicado por 0.
5. Imagina una expresión con la que tu versión del programa modificado siga comportándose erróneamente; y vuelve a modificarlo para que también pueda superar ese obstáculo.
6. Como si fueras ordenador, VAYA al punto 5.



*Hay programas que hacen cosas por sí mismos, y hay programas que te ayudan a escribir otros programas. Estos últimos se denominan **utensilios**. Por ejemplo:*

12 Renumerando líneas

Tener que aderezar las líneas de un programa, puede ser una faena tediosa, que al final nadie hace... **a no ser que** dispongan de un utensilio de programación para hacerlo (también se llaman programas de utilidad, pero todos los programas tienen su utilidad). Puedes comprar programas que renumeran líneas o puedes escribirlos tú mismo ahorrándote algo de dinero, posiblemente con la contrapartida de tener algo menos versátil.

El programa que te presento es un compromiso. Se ha escrito sobre la base que éste no es un libro de Código Máquina, por lo que el programa tiene que estar en BASIC; y dado que BASIC es inherentemente lento, el programa tiene que ser lo bastante rápido como para que realmente se use. Eso, a su vez, significa que tiene que ser bastante rudimentario. En especial, **sólo** renumera las líneas: **no** renumera automáticamente los números que son destino en instrucciones de ida, o de ida y vuelta (GO TO o GO SUB). Ya te contaré algo sobre esto, después de examinar el programa.

NUMEROS DE LINEA EN LOS PROGRAMAS

Igual que la mayoría de los utensilios, éste requiere también saber lo que pasa en las entrañas del Spectrum cuando se pulsan sus teclas. Recordarás (Fácil Programación pág. 93) que tu programa queda grabado en la memoria de lectura-escritura como una serie de caracteres|octetos, comenzando a partir de la dirección señalada por la variable sistemal PROG y terminando inmediatamente antes de la dirección señalada por VARS. Consultando el Manual, descubrirás que puedes hallar esas direcciones mediante los comandos:

PROG: PEEK 23635 + 256 * PEEK 23636

VARS: PEEK 23627 + 256 * PEEK 23628

También puedes descubrir que cada línea en un programa está grabada según el formato:

NS	NJ	LJ	LS	codificación para esas instrucciones	ENTER

siendo NS y NJ los octetos senior y junior correspondientes al número de línea, y LJ y LS los octetos junior y senior que indican el total de caracteres que hay en esa línea.

Específicamente, si el número de línea es n, tenemos que

NS = INT (n/256)

NJ = n - 256 * NS

e igualmente, es válido para LJ y LS. Así para la línea 7ØØ, tendremos:

NS = INT (7ØØ/256) = 2

NJ = 7ØØ - 256 * 2 = 188



y esos son los primeros dos octetos grabados en la celdilla de memoria en que se encuentra la línea 7ØØ.

Si cambiamos NJ, digamos a 198, engañamos al sistema operativo, haciéndole pensar que esa línea es realmente la línea 71Ø (= 2 * 256 + 198). Eso nos sugiere cómo renumerar las líneas.

Escrutamos todo el área del programa, examinando cada vez que aparece el carácter ENTER (cuyo código es 13). Una vez que encontremos uno, sabemos que los siguientes dos octetos son el número de línea. HINCAMOS ahí el valor que queramos.

PRIMER INTENTO

Si escribes una rutina que simplemente haga eso, te toparás con algunos peros. Primero, fracasa en la renumeración de la primera línea, porque no está detrás de ningún código 13. En segundo lugar (y no es fácil de ver por dónde voy) si sucede que uno u otro de los octetos LS, LJ es 13 ¿qué ocurre?

Estos fallos se remedian fácilmente: renumerar la primera línea como curso de acción, y saltarse los octetos correspondientes a LJ y LS.

Supongamos por el momento, que quieres que los nuevos números comiencen a partir de 1Ø y vayan de 1Ø en 1Ø. Eso nos llevaría a un programa más o menos así:

```
1000 LET prog=PEEK 23635+256*PEEK 23636
1010 LET vars=PEEK 23627+256*PEEK 23628
1020 LET ns=0: LET nj=10
1030 FOR i=prog TO vars-1
1040 IF i=prog THEN  POKE i,ns: POKE i+1,nj:
     LET i=i+4: LET nj=nj+10: IF nj>=256
     THEN  LET nj=nj-256: LET ns=ns+1
1050 IF PEEK i=13 THEN  POKE i+1,ns: POKE i+2,nj:
     LET i=i+5: LET nj=nj+10: IF nj>=256
     THEN  LET nj=nj-256: LET ns=ns+1
1060 NEXT i
```

Ahora teclea esto; precedidos por algunos números de línea, como comprobación del programa:

```
 1   REM
 2   REM xxxx
17   REM
```

y demás. Mándale que vaya a la línea 1ØØØ.

Lo siento querido: te tropiezas con un mensaje de error:

N Statement Lost, 1Ø6Ø: 1

Desde luego, has perdido una instrucción, la 1Ø6Ø, pero ¿por qué?

A listar y a trabajar duramente.

SEGUNTO INTENTO

Desde luego... este programa idiota acaba renumerandose **a sí mismo.** Una vez que haya renumerado la 1Ø3Ø, la instrucción en 1Ø6Ø le dice que otra "**i**", con lo que vuelve a la 1Ø6Ø que ya no existe más...



Bien; también se remedia fácilmente: cesa de renumerar **antes** de llegar a la propia rutina de renumeración. La manera fácil es cambiar la línea 1Ø3Ø sustituyendo **vars-1** con **vars-lon**, siendo **lon** la longitud de la rutina de renumeración en octetos (lo cual determinas usando **vars-prog**). Con esta rutina particular, lon = 385, por lo que la línea 1Ø3Ø sería:

```
1Ø3Ø  FOR i = prog TO vars - 385
```

y ahora funciona. Y para ser BASIC, relativamente rápido. Tarda unos 2Ø segundos en renumerar un programa de 5Ø líneas, lo cual no es instantáneo pero sí bastante rápido como para ahorrarte un montón de trabajo. (Para una rutina en Código Máquina simple e igualmente limitada, que **sí** es instantánea, véase Código Máquina y Mejor Basic pág. 159, o Código Máquina del Spectrum Capítulo 16).

TERCER INTENTO

Ahora bien, puede que se te antoje que posiblemente estés siendo un poquito obtuso. Esos octetos LS y LJ que marcan la longitud de la línea, son explotables: en lugar de ir escrutando laboriosamente hasta encontrar el código 13, podríamos saltar inmediatamente a los siguientes octetos que indican el número de línea, si añadiéramos la longitud de la línea (que es sumar o restar un octeto o dos!).

Puede que se te ocurra que quizás no se ahorre mucho tiempo a causa del tratamiento requerido para la suma y demás. La única manera de determinarlo es intentarlo.

Aquí hay un programa siguiendo esas directrices: y claramente tiene una ventaja -es más breve.

```
1010 LET i=PEEK 23635+256*PEEK 23636:
     LET ns=0: LET nj=10:
     LET fin=PEEK 23627+256*PEEK 23628-285
1020 IF i>=fin THEN  STOP
1030 POKE i,ns: POKE i+1,nj:
     LET nj=nj+10:
     IF nj>=256 THEN  LET nj=nj-256: LET ns=ns+1
1040 LET i=i+PEEK (i+2)+256*PEEK (i+3)+4:
     GO TO 1020
```

La variable **fin** es desde luego la vieja variable **vars,** menos la longitud (285) de esta rutina.

Para ver cuál es el método más rápido, las cargaremos en un programa suficientemente largo, usando el comando MERGE, y cronometraremos la ejecución de la renumeración (necesitarás programas con números de línea menor de 1ØØØ para evitar volver a escribir parte de la rutina; por lo tanto: véase a continuación). Haz esto antes de ver el resultado o como mínimo, haz una aproximación lógica...

En mi pasada de prueba, con un programa de 5Ø líneas, los tiempos logrados fueron:

Primera rutina:	2Ø segundos
Segunda rutina:	1 segundo

Esta mejora ulterior con resultados impresionantes, demuestra cuán importante es no parar de pensar sobre un programa, simplemente porque **funcione.**



VERSION DEFINITIVA (?)

Pero, ¿todavía no hemos acabado?. La tarea final es refinar la rutina para maximizar su utilidad. ¿Qué criterios debemos satisfacer?

1. La rutina deberá ocupar tan poca memoria como sea posible.
2. Deberá escribirse en tan pocas líneas como podamos.
3. Esas líneas debieran colocarse en alguna parte donde raramente, si alguna vez, se usan en un programa normal. Fallando lo del Código Máquina y lo de bajar RAMTOP, el lugar donde ponerlas es en las líneas 9995-9998. (Exceptuando el 9999 para aquellos casos en que atacas una ampliación del programa, usando 9999 VAYA a donde sea...)
4. Los nombres escogidos para las variables, deben ser de los que no usas normalmente, de manera que siempre pueda mezclarse la rutina en otro programa con cierta garantía de que no haya conflicto de variables.
5. Sería bonito comenzar y acabar en líneas arbitrariamente elegidas, y poder señalar incrementos y valores de comienzo de los nuevos números que también fueran arbitrarios (lo primero exigiría comprobaciones extras que costarán tiempo, y yo no lo incluiría. Lo segundo es esencial: puede que quieras incluir una de tus rutinas favoritas en un programa y haya solape en los números de línea: por eso primeramente quisiera renumerar la rutina a agregar).

Teniendo este criterio en cuenta (y observando que no es posible lo de todo para todos; que hay conflictos que exigen compromisos) yo terminaría con algo como:

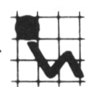
```
9994 INPUT 'Ini,paso';o1,o2:
     LET o3=INT (o1/256): LET o4=o1-256*o3
9995 LET o=PEEK 23635+256*PEEK 23636:
     LET oo=PEEK 23627+256*PEEK 23628-315
9996 IF o>=oo THEN  STOP
9997 POKE o,o3: POKE o+1,o4: LET o4=o4+o2:
     IF o4>=256 THEN  LET o4=o4-256: LET o3=o3+1
9998 LET o=o+PEEK (o+2)+256*PEEK (o+3)+4:
     GO TO 9996
```

Esta tardó un poquito más -unos 3 segundos en un programa de 5Ø líneas. Es el precio pagado por un poquito más de flexibilidad.

La razón de todas esas o's, es que, desde luego, apenas nunca uso "o" para variables en los programas normales, porque se las confunde siempre con el "cero".

Esta es una rutina genuinamente utensilio. La idea es conservarla en cinta con un nombre como "renum", y antes de comenzar a escribir un programa, cargarla en la parte superior del área de programa (por eso tiene los números de línea tan altos). Con ella escribir tu programa y maquillar sus números de línea, llamando a la rutina **renum**, usando SALTO al 9994; arreglar los GO TOs y GO SUBs manualmente; y cuando estés satisfecho con el programa, quitar las líneas 9994-9998 y guardar el programa final.

Si has olvidado cargar **renum** al empezar, siempre puedes usar el comando MERGE posteriormente para congregar la rutina con tu programa.



RENUMERACION DE BLOQUES

Este programa está bien si quieres producir listados que vayan 1Ø, 2Ø, 3Ø... siempre y sin vanos -ó 1Ø2, 1Ø4, 1Ø6,... para ser diferentes- pero eso no es siempre lo que deseas. Desde luego, maquilla el listado; pero puede que lo haga menos aprovechable.

Si has leído las páginas 87-92 de Fácil Programación, sobre buen estilo, sabrás que una forma de producir programas "civilizadamente" es desglosarlo en bloques, siendo cada bloque una subrutina. Usar nombres de bloques en lugar de números de línea (tales como LET bloque = 1ØØØ); y comenzar cada bloque sobre un número de línea redondo (1ØØØ, 2ØØØ, etc. dependiendo de los bloques con los que tengas que tratar).

Al renumerar desde el principio hasta el final, de alguna manera distorsionamos esta estructura tan cuidadosamente producida. Por otro lado, cuando estás desarrollando el bloque 1ØØØ te encuentras rápidamente que estás quitando las pifias en líneas como 1Ø35, y posteriormente en la 1Ø32 y en la 1Ø33, etc.; y o bien terminas todo con mala apariencia; o puede que incluso llegues a necesitar insertar una línea entre el 1Ø46 y el 1Ø47, y ya sabes que al intérprete BASIC no le gusta ni el 1Ø46$^{1}/_{2}$ ni el 1Ø46,5.

De forma que lo bueno sería renumerar **dentro de un bloque** (donde, aprovecho para decirte, que los problemas de SALTOS son menores, y si el programa está bien estructurado apenas ni existen). La siguiente rutina hace esto: le das las líneas de principio y fin de bloque, el nuevo número en que quieres que comience, y el nuevo incremento: ella hace el resto.

Si le pides que renumere un bloque a partir de un número de línea más bajo que el presente, saca el pitido y pregunta de nuevo el dato. (Esto se hace, porque el sistema BASIC no es feliz si le desordenas las líneas del programa en la memoria). De igual manera, si terminas de renumerar el bloque con un número de línea que es mayor que el siguiente,vuelve a pitar y continúa renumerando hasta el final.

La rutina tal y como está escrita, reside en la línea 9ØØØ. Si directamente tecleas el comando LET renum = 9ØØØ puedes usarla tecleando GO TO renum. Desde luego, sería preferible un número mayor de línea como 999Ø, y también deberías cambiar las variables a otras menos comunes, como hemos hecho antes: aquí las hemos evitado para conseguir más claridad en el programa.

```
8999 STOP
9000 INPUT 'Comienzo/Final de bloque';stt,end:
     INPUT 'Comienzo/Paso de numeracion';nst,inc:
     IF nst < stt THEN  BEEP .1,0: GO TO 9000
9010 IF end >= 8999 THEN  LET end=8998
9020 LET ns=INT (nst/256): LET nj=nst-256*ns
9030 LET i=PEEK 23635+256*PEEK 23636
9040 IF 256*PEEK i+PEEK (i+1)<stt
      THEN  GO SUB 9080: GO TO 9040
9050 IF 256*PEEK i+PEEK (i+1)>end
     THEN  GO TO 9090
9060 POKE i,ns: POKE i+1,nj: LET nj=nj+inc:
     IF nj>=256 THEN  LET nj=nj-256: LET ns=ns+1
9070 GO SUB 9080: GO TO 9050
9080 LET i=i+PEEK (i+2)+256*PEEK (i+3)+4: RETURN
9090 IF 256*ns+nj-inc>256*PEEK i+PEEK (i+1)
     THEN  BEEP .1,20: LET end=8998: GO TO 9060
```

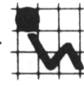



También éste es un utensilio práctico, especialmente al congregar programas y rutinas usando el comando MERGE.

Proyecto

Piensa un poco sobre cómo ocuparse automáticamente de la renumeración de los saltos GO TO y GO SUB. (No hemos hablado mucho de esto, en parte debido a la pereza, y en parte a que el resultado es muchísimo más lento, y principalmente, porque usando "subrutinas con nombre", que es una buena práctica en programación, es preciso pensar bastante más. Por ejemplo, en COLORERO, la subrutina **prueba** está en la línea 2ØØØ, y queda establecido a inicializar la variable prueba = 2ØØØ en la línea 2Ø. Si renumeraramos, no tendríamos que cambiar la línea 25Ø que nos desvía con vuelta, a esa rutina (GO SUB prueba); lo que tendríamos que hacer es cambiar en la línea 2Ø el valor que hemos dado a la variable **prueba.** E incluso aunque te preocupes de esto, hay percances con los saltos a subrutinas y puede terminar todo embrollado).

Las rutinas no están protegidas contra programas que usen números altos de línea (por encima de 9999). Hazlo. (Lo hará todo un poco más lento, pero ¿merece la pena?).



No sólo es simplemente hermoso
-también es hermosamente simple.

13 Polígonos

Básicamente es una idea simple y directa, pero sólo puedes hacerla por tí mismo, si te encuentras a gusto con la trigonometría (SENO, COSENO, TANGENTE, y demás). El objetivo es desarrollar los gráficos del Spectrum para que permita la construcción de polígonos -y de criaturas imaginativas de la misma ralea.

Un polígono, lo que recordarás sin ninguna duda, es una figura hecha por segmentos rectos. En cierto modo, eso es todo lo que puede dibujar el Spectrum, pero si los segmentos son muy cortitos y se empalman, dan como resultado curvas. Por eso es por lo que un dibujo en alta resolución de Bo Derek no **aparenta** que su figura se obtenga de líneas rectas...

Bueno, volvamos al Spectrum. Un polígono es **regular** si todos sus lados y todos sus ángulos son iguales, por lo que es simétrico. Para dibujar un polígono regular de n lados, dentro de un círculo de radio r, cuyo centro está en x, y, usamos el programa:

```
10 INPUT 'Numero de lados?';n
20 FOR i=0 TO n
30 LET a=x+r*COS (2*i*PI/n):
   LET b=y+r*SIN (2*i*PI/n)
40 IF i=0 THEN  PLOT a,b
50 DRAW a-PEEK 23677,b-PEEK 23678
60 NEXT i
```

Tal y como está, hay peligro de salirnos de la pantalla, de forma que agregas:

```
 5 IF r>x OR r>y OR r>255-x OR r>175-y THEN  RETURN
```

He puesto la condición de REGRESO, porque estoy pensando usarlo como subrutina; por lo que también necesitaremos

```
70 RETURN
```

para que todo quede perfecto. Muy bien, pero hasta ahora no veo ningún uso para esto. Así que añade:

```
100 INPUT 'Radio?';r
110 INPUT 'Centro?';x,y
120 CLS
130 GO SUB 5
```

Ensaya así, pero recuerda comenzar yendo a la 100 en vez de ejecutar el programa.

Bien, pero pronto resulta aburrido. Sin embargo, podemos hacer los resultados mucho más bonitos usando bucles. Por ejemplo, "delecta" la línea 10 y teclea:

```
200 LET n=5: LET x=127: LET y=87
202 CLS
210 FOR r=5 TO 85 STEP 5
220 GO SUB 5
230 NEXT r
```




lo que da la figura 13.1.

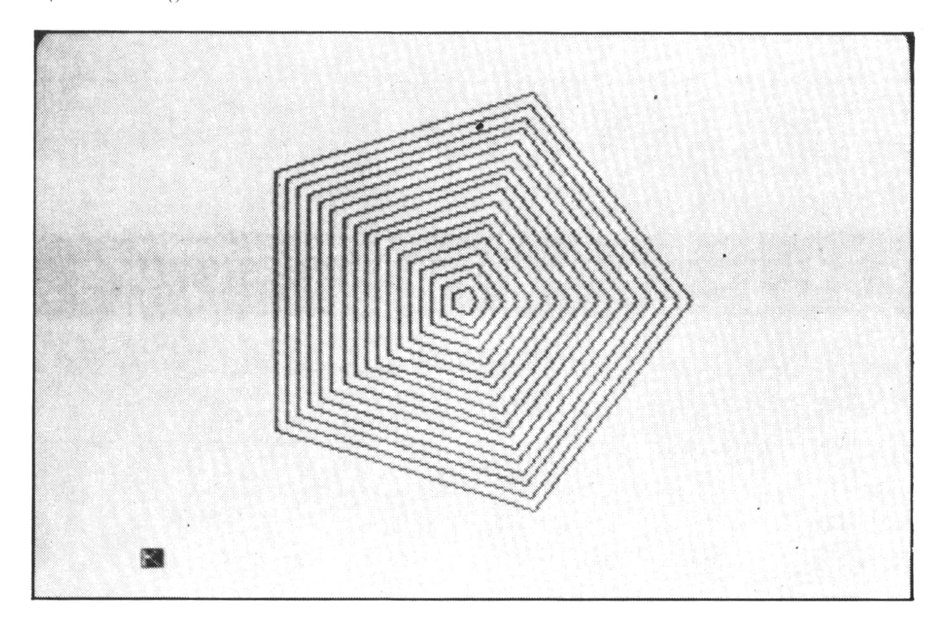


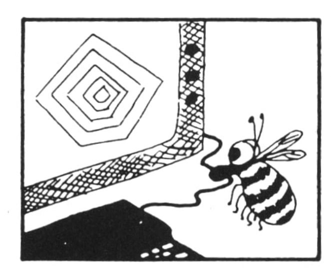
Figura 13.1 Pentágonos concéntricos.

Si éstas te gustan, quitate de encima la de HAGA n=5 en la línea 2ØØ, y añade:

```
205 INPUT 'Numero de lados?',n
```

de manera que puedas ensayar numeros diferentes. Mándale ahora que vaya a la 2ØØ.

Ensaya con n = 5Ø: son círculos bastante exactos (aunque mucho más lentos de como salen con el comando CIRCLE). Eso ès lo que quería decir con la observación sobre Bo Derek...





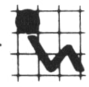
ROTACIONES

Después de un rato, incluso esto aburre. Todos esos deprimidos polígonos apuntando siempre a una misma dirección. Añadámosle rotación:

```
206 INPUT 'Rotando?';ro
```

y vamos ahora a girar todos ro grados, para lo que cambiamos la línea 3Ø para que sea:

```
30 LET a=x+r*COS (2*i*PI/n+ro*PI/180):
   LET b=y+r*SIN (2*i*PI/n+ro*PI/180)
```



Vamos a hacer cosas más imaginativas:

```
300 LET n=7: LET x=127: LET y=87
305 CLS
310 FOR r=5 TO 85 STEP 5
320 LET ro=r*2
330 GO SUB 5
340 NEXT r
```

Usa GO TO 3ØØ: te saldrá la figura 13.2. Cambia el n = 7 para variar, a un comando INPUT. Cambia el r * 2 de la línea 3Ø para que sea r, o bien r * 3, o lo que te apetezca. Incluso r * r / 5Ø es bastante bonito.

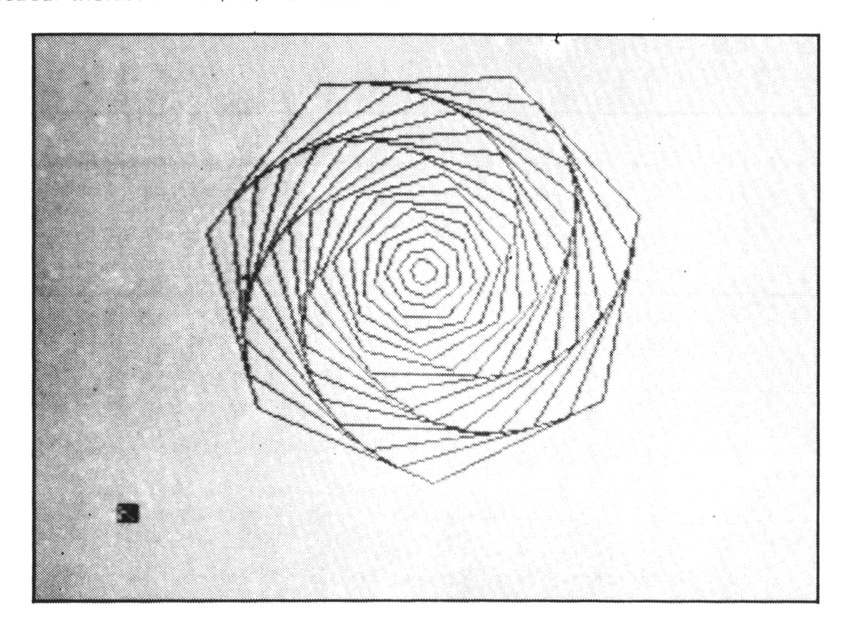


Figura 13.2 Heptágonos rotantes.

Mirando a la línea 3Ø en su presente forma, no puedo evitar el preguntarme lo que sucedería si las posiciones donde punteo, coordenadas a y b, fueran cambiadas en cantidades **diferentes.** En otras palabras, cambia la instrucción para que sea:

```
30 LET a=x+r*COS (2*i*PI/n+ro1/180):
 LET b=y+r*SIN (2*i*PI/n+ro2/180)
```

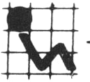
donde ro1 y ro2 se imponen por teclado usando:

```
206 INPUT 'Rotando?';ro1,ro2
```

Ensaya esto empezando por GO TO 2ØØ. Es como si se torcieran los polígonos. Ahora adapta la línea 32Ø de la rutina que comienza en 3ØØ, para que sea:

```
320 LET ro1=r*2: LET ro2=r*3
```

La cosa se esta complicando ahora, y los dibujos se están volviendo menos predecibles.



LADOS CURVOS

¿Y ahora qué más? Bien, podemos usar el comando de dibujar para hacer curvas, así como para hacer rectas (Fácil Programación pág. 34). Lo podemos añadir como una opción; cambia la línea 5Ø a:

```
50 DRAW a-PEEK 23677,b-PEEK 23678,curva
```

y arréglatelas para que de alguna manera se imponga por teclado la magnitud de la curva, tal v como con:

```
207 INPUT "curva?";curva
```

Verás que curvando entre -4 y +4 aproximadamente, funciona mejor; y yo prefiero los valores negativos.

Experimenta durante un rato con GO TO 2ØØ. Ahora pón las curvas dentro del bucle de la línea 3ØØ. Por ejemplo, añade:

```
315 LET curva=-r/25
```

y cambia la 32Ø a

```
320 LET ro1=r/2: LET ro2=r/3
```

Otra vez se está complicando demasiado, ahora...

ESTRELLAS

Si cambias la línea 2Ø para que (digamos)

```
20 FOR i=0 TO 2*n
```

puedas imponer valores a n = 5/2, 7/2, etc. obtendrás una estrella de 5 o de 7 puntas. Con

```
20 FOR i=0 TO 3*n
```

puedes ensayar n = 5/3, 7/3, 8/3, etc.; y en general con

```
20 FOR i=0 TO k*n
```

valores de la forma n = (numero entero)/k producen criaturas con forma de estrella. (Necesitas imponer el valor de k por teclado o asignarlo por instrucción).

Puedes añadir comandos de color; usando OVER 1... La variedad es infinita. Aquí hay uno bastante bonito para terminar: tecléalo y empieza con GO TO 4ØØ. La figura 13.3 muestra el resultado.

Cambia primero la línea 2Ø de la subrutina a:

```
20 FOR i=0 TO 3*n
```

y luego añade:

```
400 LET n=11/3: LET x=127: LET y=87
405 PAPER 0: BORDER 0: OVER 1: CLS
410 LET ro1=0: LET ro2=0
415 FOR r=5 TO 75 STEP 10
418 LET curva=-r/25
420 INK 1+r/16
430 GO SUB 5
440 NEXT r
```



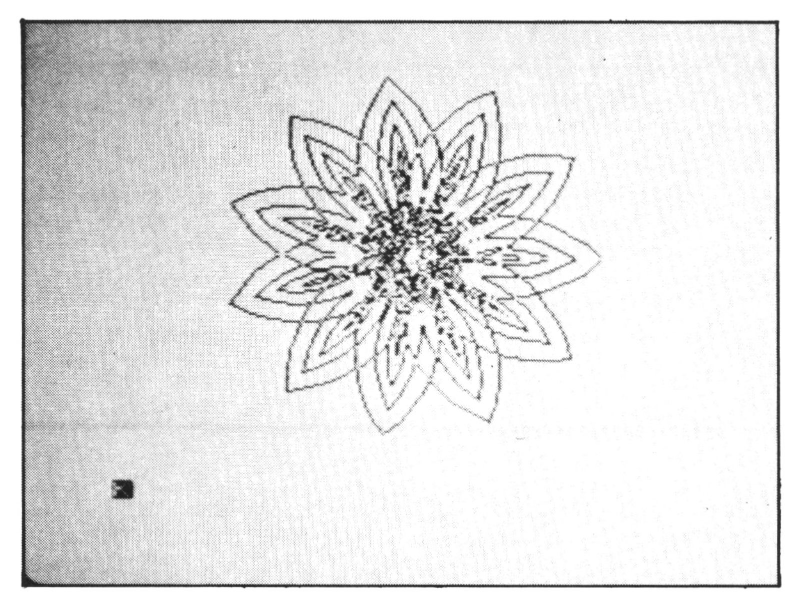


Figura 13.3 Lados curvos $5^1/_2$-gones.

Obviamente puedes usar estas rutinas dentro de otros programas. Por ejemplo, no hemos intentado en absoluto cambiar el centro x, y. Mira a ver si puedes dibujar ringlas 4 x 4 de pentágono; una línea que solape heptágonos; una línea curva de hexágonos que crece más y más y **también** están girados; un trazado aleatorio de polígonos de lados aleatorios coloreados aleatoriamente y mezclados aleatoriamente con estrellas...

Pero si has seguido conmigo hasta este punto, no creo que necesites que te urja para que hagas cosas por tí mismo.



Bajó a la cripta, encontró el documento,
lo abrió, y tuvo que recurrir a la...

14 Criptografía y Criptoanálisis

El arte de cifrar palabras y conseguir códigos, y de descifrar o decodificar los resultados sin saber cuál es la tabla de conversión de códigos.

Es un triste comentario sobre la condición humana, que el uso más temprano que se recuerda de mensajes codificados, fue para evitar interceptación por el enemigo, y se remonta a los lacedemónicos en el año 400 a.d.C. Ya el primer libro sobre el tema fue **En Defensa de las Fortificaciones** y lo escribió Tacticus en el siglo cuarto a.d.C.

El problema de la conversión -decodificar un mensaje sin saber el código- tiene mayor importancia que simplemente la militar. Los historiadores necesitan saber los mensajes que se pasaron entre los comandantes durante la Guerra Civil Americana; y los lingüistas, descifrar manuscritos antiguos, tales como los jeroglíficos egipcios o los "lineales" de Micenas. Alguno de éstos puede que no estuvieran codificados **entonces,** pero sí lo están **ahora.**

El ordenador puede ser un instrumento muy potente para el criptoanálisis, porque puede desarrollar "aclaratorio" con ensayos y errores a gran velocidad. (Los ensayos y errores que no sean aclaratorios ocupan demasiado tiempo, incluso en el Cray-1. Tu Spectrum no soportaría una pasada).

Los códigos más simples, son **códigos de sustitución,** en que cada letra del mensaje original se convierte a otra letra de un alfabeto arbitrariamente desordenado, según la posición; para mantener este capítulo dentro de sus límites, nos concentraremos sobre esto.

CODIGOS DE SUSTITUCION

Por ejemplo, supón que el mensaje es:

"Mi carro me lo robaron"

y que el código está definido por:

```
abcdefghijklmnopqrstuvxyz
zvetrsnbjpkcuxdfgayohlimw
```

Por tanto, el mensaje codificado sería:

"uj ezaad ur cd advzadx"

para lo que basta mirar en el primer renglón y sustituir por la equivalente en el segundo.

El siguiente programa recoge un mensaje y lo codifica o cifra, mediante un código de sustitución aleatorio. Posteriormente, lo desarrollaremos hasta que sea una rutina que decodifique mensajes de esos, suponiendo que el usuario es inteligente.

```
100 LET a$='abcdefghijklmnopqrstuvwxyz'
105 PRINT a$
110 LET b$=''
120 FOR i=1 TO 26
130 LET m=INT (1+(27-i)*RND)
140 LET b$=b$+a$(m): LET a$=a$( TO m-1)+a$(m+1 TO )
150 NEXT i
```



```
160 PRINT b$
```

Simplemente, lo que hace es alterar aleatoriamente el orden del alfabeto, eligiendo al azar la primera letra, luego la segunda, también aleatoriamente y sacada de las que quedan, y así sucesivamente. Estúdialo cuidadosamente, las revistas están llenas de rutinas de "aleatorización", que pinzan dos letras al azar, las canjean entre sí, y lo repiten un montón de veces, lo que es increíble para lograr este resultado!

Ahora, para poder imponer el mensaje a codificar:

```
200 INPUT m$
210 PRINT m$
215 LET c$=""
220 FOR i=1 TO LEN m$
225 LET c=CODE m$(i)
230 IF c=32 THEN  GO TO 250
235 IF c<97 THEN  LET c=c+32
240 IF c>=97 AND c<128 THEN  LET c$=c$+b$(c-96)
250 NEXT i
260 PRINT c$
```

Las observaciones a hacer son que en la línea 235 se convierten las mayúsculas en minúsculas; la 23Ø detecta los blancos; y la c - 96 en la línea 24Ø aparece porque el alfabeto ocupa los códigos desde 97 a 123.

ANALISIS DE FRECUENCIA

¿Cómo un criptoanalista aborda el problema de codificación? (Desde luego, en la práctica, este método de cifrar mensajes se descubre tan fácilmente que no tendrá ninguna utilización práctica: nuestra tarea inmediata es ver por qué).

Lo esencial a tener en cuenta es que un idioma normal, no utiliza con la misma frecuencia las letras de su alfabeto. La letra "E", por ejemplo, se empiea mucho más frecuentemente que la letra "Z". A continuación hay una tabla del promedio de aparición, determinado al analizar telegramas oficiales, (y por supuesto en inglés). Con cifras para 1ØØ letras.

a	7.4	n	7.9
b	1.Ø	o	7.5
c	3.1	p	2.7
d	4.2	q	Ø.3
e	13.Ø	r	7.6
f	2.8	s	6.1
g	1.6	t	9.2
h	3.4	u	2.6
i	7.4	v	1.5
j	Ø.2	w	1.6
k	Ø.3	x	Ø.5
l	3.6	y	1.9
m	2.5	z	Ø.1

En otras palabras, la letra más común es "e" que aparece un 13% de las veces; luego la "t" con un 9,2%; luego "n", "r", "o", "a", "i", "s", "d", después de las cuales, la frecuencia de aparición es de menos de 4%. De forma que es una buena estimación suponer que la letra que más veces aparezca en la versión cifrada del mensaje, suficientemente largo, debe ser la que corresponde a la "e", y así sucesivamente.



Para el mensaje anterior (que es bastante corto) las frecuencias son:

a	4	de 18
c	1	
d	4	
e	1	
j	1	
r	1	
u	2	
v	1	
x	1	
x	2	

Las más comunes son "a" y "d", que corresponden a "r" y "o" respectivamente; no es de mucha ayuda, pero es que el mensaje es muy corto. Si comenzáramos con un texto más largo, tal y como:

"Fisgando e hincando datos en sus sedes de la computación"
es un excelente libro de computación

su versión cifrada, sería:

"sjynžxtd r bjxezxtd tzody rx yhy yrtry tr cz urudajz
ry hx riercrxor cjvad tr edufhozejdx

(y en la práctica, los blancos serían omitidos). Las frecuencias serían entonces:

a	2
b	1
c	3
d	7
e	4
f	1
h	3
i	1
j	5
n	1
o	3
r	12
s	1
t	6
u	3
v	1
x	7
y	7
z	6

Una estimación de la frecuencia de cada letra en el idioma de los hispano-parlantes, es:

a	12,5	k	0,0	t	5,0
b	1,1	l	5,6	u	4,8
c	2,8	m	3,4	v	0,8
d	5,8	n	7,4	w	0,0
e	12,1	ñ	0,0	x	0,0
f	0,7	o	8,8	y	1,2
g	1,3	p	2,7	z	0,4
h	0,6	q	0,7		
i	6,5	r	5,8		
j	0,1	s	7,1		

La letra que más veces aparece es la "r", que supondremos, corresponde a la "e" o a la "a", y con eso ensayaremos; luego vienen la "d", la "x" y la "y", seguidas muy de cerca por la "t" y la "z", que si se mantuvieran las probabilidades, corresponderían a la "c", a la "o", a la "m", a la "f", a la "r". Eso nos permite ensayar y comprobar para ver lo que obtenemos.

Desde luego, que la más probable ha de corresponder a la "e", en cuyo caso el mensaje sería:

.... e .... e. ... .e.e. .e .. .e.....
e. .. e..e.e..e ..... .e ....

Pero si la "r" correspondiera a la "a" en este caso, sería:

.... a .... a. ... .a.a. .a .. .a.....
a. .. a..a.a..a ..... .a ....



y en este caso, que contamos con la ayuda de tener las palabras separadas por blancos, podemos ver que la "r" corresponde en realidad a la "e" y además, poder detectar partículas pequeñas como "d", "e", "en".

En otros casos, en que no estén las palabras separadas por blancos, será más difícil empezar a hacer ensayos. Pero siempre ha de tenerse en cuenta, que la frecuencia con que aparecen en una determinada frase, se acerca a la probabilidad general calculada anteriormente, pero no tiene por qué coincidir.

EL PROGRAMA

Por tanto, un programa de criptoanálisis para códigos de sustitución, deberá presentar la cuenta de las veces que aparecen cada una de las diversas letras; a partir de esa cuenta, ensayarás con distintas hipótesis y verás cuál es el resultado. Empecemos con la cuenta, mediante el siguiente programa:

```
500 REM analisis de frecuencia
510 DIM n(26)
515 LET col=0
520 LET tot=LEN c$
525 FOR i=1 TO 26
530 FOR j=1 TO tot
540 IF CODE c$(j)-96=i THEN  LET n(i)=n(i)+1
550 NEXT j
554 LET s$=STR$ (.01*INT (100*n(i)/tot)):
     IF s$(1)='.' THEN  LET s$='0'+s$
555 PRINT TAB col;CHR$ (i+96);' ';s$;
560 LET col=col+8
570 IF col=32 THEN  LET col=0
580 NEXT i
```

Así contamos las veces que aparece. Las líneas 555 a 58Ø producen un listado en cuatro columnas. La línea 554 es un intento (fructífero) de producir solamente dos decimales en el número. Si se omite la segunda parte en relación con s$ (1) te encontrarás con que algunos números aparecen impresos como

.Ø9

mientras que otros lo hacen como

Ø34

lo que da sensación de desaliñado. Colocando un cero delante, arregla este inconveniente. Pero, realmente, puedes acusarme de pedantería.

Ahora viene lo correspondiente a los ensayos y comprobaciones:

```
1000 REM decodificacion
1010 LET p$='': FOR i=1 TO tot: LET p$=p$+'.': NEXT i
1020 PRINT AT 15,0;p$
1100 REM ensayo
1110 INPUT 'letra que aparece ?';k$:
      IF LEN k$<>1 THEN  GO TO 1110
1115 IF k$='0' THEN  GO TO 2500
1120 INPUT 'ensayo con letra ?';g$:
      IF LEN g$<>1 THEN  GO TO 1120
1130 FOR i=1 TO tot: IF c$(i)=k$ THEN  LET p$(i)=g$
1135 NEXT i
1140 PRINT AT 15,0;p$
1150 GO TO 1110
```




No está mal, pero tiene algunos "peros". Puedes intentar usar la "misma" letra como código correspondiente a diferentes letras en el mensaje cifrado, y no darte cuenta: de esa forma vamos al desastre. Por lo que sería bonito comprobar también esto. Para hacerlo, necesitamos conservar lo que conocemos hasta este momento.

```
  10 DIM d(26)
1125 GO TO 1500
1500 REM conservacion de lo decidido
1510 LET k=CODE k$-96: LET g=CODE g$-96
1520 FOR a=1 TO 26
1525 IF d(a)<>g OR d(a)=k THEN  GO TO 1560
1530 INPUT 'Has usado□';CHR$ (g+96);
     '□como codigo correspondiente a□';CHR$ (a+96);
     '□quieres mantenerlo?';y$
1540 IF y$='s' THEN  GO TO 1110
1550 LET d(a)=0
1555 GO SUB 2000
1560 NEXT a
1570 NEXT i
1600 LET d(k)=g
1610 GO TO 1130


2000 FOR b=1 TO tot
2010 IF p$(b)=g$ THEN  LET p$(b)='.'
2020 NEXT b
2030 RETURN
```

Todo lo que necesitamos ahora, es una manera de salir, adecuadamente informados, cuando creemos que ya hemos abierto la brecha:

```
1115 IF k$='0' THEN  GO TO 2500
```

Eso nos permite imponer "Ø" cuando se nos pregunta "¿letra que aparece?" para que nos resuma todo.

```
2500 REM resumen hasta ahora
2510 CLS
2520 PRINT 'Mensaje cifrado:''c$'''Mensaje decodificado:''p$'''
2530 PRINT 'Tabla de codigos conocidos:'
2540 FOR i=1 TO 26
2550 PRINT AT 12,i+3;CHR$ (i+96)
2560 IF d(i)<>0 THEN  PRINT AT 13,i+3;CHR$ (d(i)+96)
2570 NEXT i
2580 GO TO 1110
```

Para evitar los engaños, debes ahora suprimir las líneas 1Ø5, 16Ø y 21Ø. Luego hacer que algún otro imponga el mensaje. O bien...



PROBLEMA

Aquí hay cuatro mensajes para que los decodifiques (las respuestas están en la última página, antes de los Apéndices). Todas son citas muy conocidas, pero cada una con un diferente código de sustitución.

1. xdryrcbduvarrcghrbzvczyjxdrccrxnhzprzoazlrytrcbduvar

2. oukjdlbooxtmkmletpmrjmmwmuehmbmwtotjrjuj
   xybjluodmxnowpwxozpwrjyobejrjmlpxvmbuoxx
   jupvejwvjwouewxtbpdowtjroubmhjwmdeowtjujzevj

3. rlswdbrlyshnscarxurramfwcbrdbfdfheqaflwr
   nsfldqbqxurrhrazugubfdurnqpspsbrhqwcsgfr
   yfafesyfebqgfnfhnrdurnqnrldareqbgfnqlluldflscsasnqnrl

4. xzrkopqzrfblbkrdrkmjphpyjfbtkpfjbhireorz
   ejfplrmripkborelrdhrlriberdbxdpydrkjcbir
   eorxrdkpebhjvblb

Proyecto

1. Modifica el programa para que exponga la tabla de frecuencias de las letras si se le exige (digamos, contestanto "1" cuando te pregunta por letra que aparece.
2. Añade un ordenamiento por burbujas (Fácil Programación, pág. 65) para que enumere las letras usadas en el mensaje según el número de veces que aparece. Eso facilita la decodificación.
3. Añade otra opción (tecleando "2" ante la pregunta "¿letra que aparece?") para que presente la tabla de frecuencias habituales de las letras, para que la puedas consultar.
4. Prepara un programa para contar las combinaciones de dos letras (bigrafismos) que sean más frecuentes en hispano. (En inglés son:

en	.111	on	.077
re	.098	in	.075
er	.087	te	.071
nt	.082	an	.064
th	0.78	or	.064

   Para poder sacar ventaja de esta información extra.
5. Escribe programas que funcionen con otros tipos de códigos (hay buenas referencias en la Enciclopedia Británica y Los Descifradores de Códigos por D. Kahn) y para permitirte intentar descifrarlos.



¿Necesitas más de 23 caracteres definibles por el usuario? Ahora puedes tener 256 con un solo POKE.

15 Cambiando el Repertorio de Símbolos

Ya mencioné en el capítulo 6 que puedes establecer nuevos símbolos por encima de RAMTOP y emplearlos "hincando" un valor en la variable sistemal CHARS. Este capítulo describe el proceso en detalle.

Supongamos, para simplificar que queremos 64 nuevos símbolos. Por tanto, necesitaremos 64 * 8 = 512 octetos de espacio libre. RAMTOP apunta normalmente a la dirección 32599 en un Spectrum de 16K, de manera que tendremos que bajar ese valor a 32599 - 512 = 32087, con lo que dejaremos un **ático** de 512 octetos de superficie. Para hacerlo, teclea (directamente)

```
CLEAR 32087
```

El área cedida comienza en la dirección 32088. Que ya sabes, equivale a 125 * 256 + 88, por lo que el octeto junior es 88 y el senior 125. Podemos engañar al sistema para que CHARS indique esta nueva área, restando uno más del octeto senior (recuerda, que CHARS contiene 256 **menos** que la dirección de la tabla de caracteres). Por lo tanto, el comando real es:

```
POKE 23606, 88: POKE  23607, 124
```

Pero todavía no lo impongas.

Este programa te permite establecer 64 nuevos símbolos, y comprobarlos para asegurarte que están bien.

```
  10 FOR i=32 TO 96
  20 GO SUB 400
  30 PRINT i-31,
  40 GO SUB 200
  50 PRINT CHR$ i
  60 PRINT '
  70 NEXT i
  80 GO SUB 400
  90 STOP
 200 REM neva direccion en CHARS
 205 REM 93 y 248 para 48k
 210 POKE 23606,88: POKE 23607,124
 220 RETURN
 400 REM direccion normal en CHARS
 410 POKE 23606,0: POKE 23607,60
 420 RETURN
1000 REM 'proce' que impone el nuevo repertorio
1005 REM 63837 para 48k
1010 LET i=32088
1020 INPUT j
1030 PRINT j;'□';
1040 POKE i,j
1050 LET i=i+1: GO TO 1020
```

En el programa, la línea 200 hace que CHARS señale al nuevo área, la 400 lo pone para la posición normal; y en 1000 es la rutina que impone el repertorio de símbolos.



Comienza con el comando GO TO 1ØØØ. Como prueba, solamente usaremos seis caracteres. Exactamente igual que con los gráficos definidos por el usuario, necesitas dibujar la forma del símbolo en una retícula de 8 x 8; convertir los renglones a un valor binario con ceros y unos; y luego imponer ese valor (véase **Fácil Programación**). Aquí usaremos los seis caracteres de la figura 15.1, lo que significa que tienes que teclear sucesivamente:

255	255	255	255	255	255	255	255	(para ■)
255	129	129	129	129	129	129	255	(para □)
1	3	7	15	31	63	127	255	(para ◢)
255	255	195	195	195	195	255	255	(para ◘)
1	2	4	8	16	32	64	128	(para /)
24	126	126	255	255	126	126	24	(para ●)

Eso es suficiente: mándale parar y luego rular. Verás los seis caracteres enumerados como 1, 2, 3, 4, 5, 6 en la pantalla. Si no, compruébalo cuidadosamente

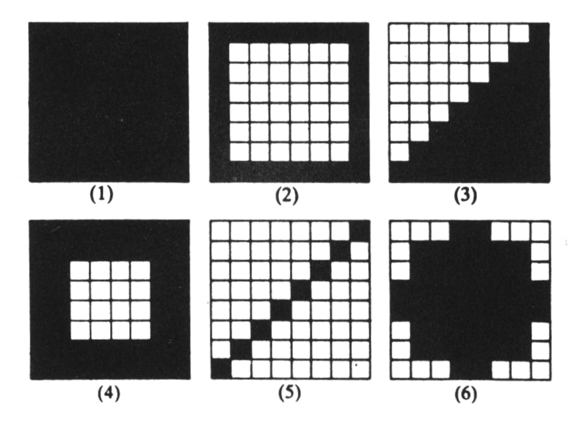


Figura 15.1 Un grupo de seis caracteres

Habiendo visto que el método funciona, la tarea final es imponer realmente esos 64 caracteres. Lo cual se hace en etapas:

1. **Diseñarlos.** Puedes usar el constructor de tipos, una de las rutinas utensilio de **Fácil Programación.**
2. Calcular los datos correspondientes a los renglones (y esa canción ya te la sabes).
3. Imponer los datos por teclado, como has hecho anteriormente.



Alpha Beta Gamma Delta

Epsilon Zeta Eta Theta

Iota Kappa Lambda Mu

Nu Xi Omicron Pi

Rho Sigma Tau Upsilon

Phi Chi Psi Omega

Figura 15.2 Diseño de un alfabeto griego.



Estoy seguro que tu mente ya está imaginando un montón de trucos ingeniosos para hacer todo esto más fácil, tales como **combinar** CONSTRUCTOR DE TIPOS con el pequeño programa anterior. Dejame simplemente mencionar uno. Para la imposición directa de los renglones en **binario**, y evitar la tediosa conversión a decimal (lo que es una idiotez, dado que el Spectrum inmediatamente los vuelve a convertir a binarios) cambia la línea 1020 para que sea:

```
1020 INPUT b$
1030 LET j=VAL ("BIN "+b$)
```

Ahora puedes imponer los valores como ristras de ceros y unos, tomados directamente de la retícula 8 x 8: 0 para un blanco, 1 para un negro.

¿Qué caracteres debiera meter? Lo anterior sugiere algunos gráficos posibles. La figura 15.2 muestra un alfabeto griego completo.

USANDO EL NUEVO REPERTORIO

Para usar los nuevos símbolos en tu programa, una vez que estén en memoria, todo lo que necesitas hacer es cambiar CHARS y llamarlos por número. Si estableces CHARS como en la línea 200, luego debes exigir CHR$ 31 + n, o simplemente pedir el símbolo normal **directamente** con el código 31 + n; te saldrá el enésimo símbolo de tu nuevo repertorio. Para usar los caracteres normales, cambia de nuevo CHARS como en la línea 400.

CONSERVANDO EL NUEVO REPERTORIO

Para guardar en cinta todo tu duro trabajo, necesitas almacenarlos octeto a octeto. Si tecleas

```
SAVE "nuevoreper" CODE 32088, 512
```

consigues guardar en cinta los 512 octetos a partir de la dirección 32088, y quedan denominados "nuevoreper". (Poniendo en marcha la cinta, como es habitual para guardar programas).

Para cargarlo de nuevo en memoria, usarás

```
LOAD "nuevoreper" CODE 32088, 512
```

Ahora te indicamos una forma incluso mejor. Escribe un programa que haga la carga. Primero teclea el programa:

```
10 LOAD "nuevoreper" CODE 32088, 512
```

Ahora, guárdalo, con el nombre "nuevacarga" (por ejemplo). Usa

```
SAVE "nuevacarga" LINE 10
```

Inmediatamente después de eso, y en la misma cinta, guarda los nuevos tipos de símbolos usando SAVE...CODE como anteriormente. Ahora tenemos la cosa en dos cachos.

Rebobina, carga el programa "nuevacarga", aprieta los botones, y vigila.

El primer mensaje en pantalla será:

```
Program: nuevacarga
```

Inmediatamente que ese haya sido cargado, comienza a rular automáticamente a partir de la línea 10, debido a que así lo hemos incluido cuando lo hemos depositado en cinta. **Deja la cinta marchando** y ese programa, automáticamente cargará los octetos designados para nuevoreper (con el mensaje Bytes: nuevoreper). Eso te ahorra recordar todas esas direcciones, número y demás...

Este método nos abre un montón de posibilidades. Puedes encadenar programas uno a continuación de otro, de forma que cada uno reclame al siguiente. Siempre que seas hábil con los controles del cassette y no tengas que volver atrás, puedes hacer uso de esta idea en todas las suertes, para aumentar eficazmente la potencia de la máquina, ya que hace uso completo de la memoria extra que supone "la cassette". Los capítulos 9 y 17, sobre ficheros en cassette y Sistemas de Gestión de Ficheros, exploran esa idea dentro de un contexto provechoso.



Una de las pequeñas pejigueras con los comandos PLOT y DRAW del Spectrum, es que pueden llevarnos a mensajes de error si los puntos que queremos pintar y dibujar se salen de la pantalla. La respuesta es diseñar un utensilio para...

16 Trazado de Curvas sin encontronazos

La manera más fácil de dibujar curvas, es escribir un bucle, que después de "pintar" el punto de arranque, "dibuja" sucesivamente hasta otros puntos, generados a partir de una lista de datos o de una fórmula. Sin embargo, esto nos crea problemas si los puntos se salen de la pantalla. Lo que sigue es un recuento golpe a golpe del desarrollo de un método para evitarlo. En algunos puntos, se vuelve un poquito matemático; pero si las matemáticas no es tu punto fuerte, ignora el álgebra y concéntrate en la estructura general.

Recuerda que para los gráficos en alta resolución, el Spectrum emplea una retícula que tiene 176 renglones y 256 columnas (numeradas de Ø a 175 y de Ø a 255) comenzando a partir del córner inferior izquierdo de la pantalla. El borde de la pantalla forma por lo tanto, un rectángulo cuyo tamaño es de 176 x 256 "motas". La clave de todo el asunto es plantear una subrutina, a la que entregamos las coordenadas (x1, y1) y (x2, y2) de los dos puntos, **que no necesariamente están en la pantalla;** determinar dónde la línea que los une sufre el encontronazo con el borde de este rectángulo (Figura 16.1).

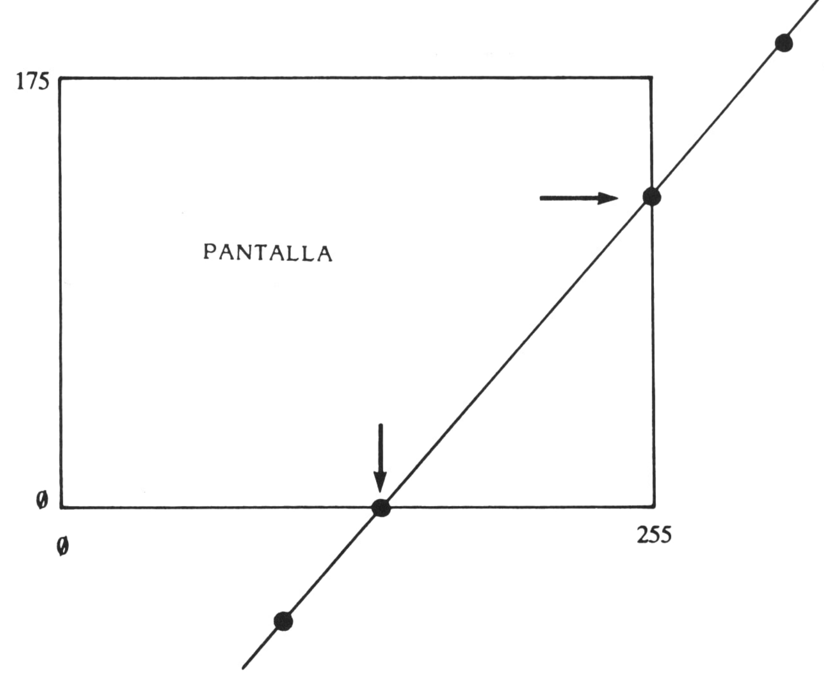


Figura 16.1 ¿Dónde una línea entre dos puntos, choca con los bordes de la pantalla?



Una pizca de la vieja geometría cartesiana, nos permite computar esos puntos. La expresión algebraica

$$\frac{(a - b) * (d - e)}{(c - b)} + e$$

aflora repetidamente en diversos disfraces, lo que es una señal de que conviene definir una función para ella (véase capítulo 3).

Usándola, conseguimos la siguiente rutina (los números de línea pueden parecer un poquito irregulares, la idea es que si continúas a través de todo ese capítulo y tecleas todas las líneas del programa al pasar, terminaras con un programa completo; pero la descripción irá subrutina por subrutina. Siempre es más fácil de teclear, y de diseñar, programas largos de esta manera).

```
   2 DEF FN a(a,b,c,d,e)=(a-b)*(d-e)/(c-b)+e
1000 REM seg
1010 DIM x(2): DIM y(2)
1020 IF x1=x2 THEN  LET xt=x1:
     LET xb=x1: LET yl=-1: LET yr=-1
1025 IF y1=y2 THEN  LET xt=-1:
     LET xb=-1: LET yl=y1: LET yr=y1
1030 IF x1<>x2 AND y1<>y2 THEN
     LET xt=FN a(175,y1,y2,x2,x1):
     LET xb=FN a(0,y1,y2,x2,x1):
     LET yl=FN a(0,x1,x2,y2,y1):
     LET yr=FN a(255,x1,x2,y2,y1)
1090 LET q=1
1100 IF xt>=0 AND xt<=255 THEN
     LET x(q)=xt: LET y(q)=175: LET q=q+1
1110 IF xb>=0 AND xb<=255 THEN
     LET x(q)=xb: LET y(q)=0: LET q=q+1
1120 IF yl>=0 AND yl<=175 THEN
     LET x(q)=0: LET y(q)=yl: LET q=q+1
1130 IF yr>=0 AND yr<=175 THEN
     LET x(q)=255: LET y(q)=yr
1140 RETURN
```

En un programa, esta rutina necesita que se le abastezca con los cuatro números x1, y1, x2, y2, (que son las coordenadas de los dos puntos): y entrega las coordenadas de los puntos donde la línea en cuestión choca con el borde del rectángulo, en las variables x (1), y (1), x (2), y (2).

Para ser capaces de emplear esta subrutina, debemos mandarle que vaya a la línea 1000, o escribiendolo en un estilo civilizado, haremos previamente

```
22 LET seg=1000
```

con lo que ya podremos mandarle que vaya a seg. Así que añade la línea 22.

Es sensato **comprobar** este bicho antes de proceder con los siguientes: el álgebra es lo suficientemente liosa como para que no detectemos en esta etapa algunos defectos, que provoquen posteriormente estragos. Así que añade **temporalmente** estas líneas:



```
100 LET x1=127: LET y1=87
110 FOR t=1 TO 50
120 LET x2=500*SIN (t*PI/50):
    LET y2=500*COS (t*PI/50)
130 GO SUB seg
140 PLOT x(1),y(1): DRAW x(2)-x(1),y(2)-y(1)
150 NEXT t
160 STOP
```

Ahora a rular: si lo que sale es un conjunto de líneas radiales desde la mitad de la pantalla, que se detienen en los bordes, todo está bien. Si no comprueba de nuevo el listado. Una vez que la rutina esté comprobada, suprime las líneas 1ØØ a 16Ø.

Con esto está todo el trabajo cerebral. La mayoría de lo que queda es mera rutina... o como mínimo, subrutina...

COMO DIBUJAR CURVAS

La primera labor es definir la estructura general. Hay dos maneras principales de dibujar una curva:

1. Un **gráfico.** Dibuja FN (t) en función de t, siendo FN una función. Véase Fácil Programación para más detalles.
2. Una **curva parametrizada.** Dibuja FNb (t) en función de FNa (t), siendo FNa y FNb **dos** funciones, en las que se denomina **parámetro** a la variable t.

Sería bonito que nuestra rutina permitiera una y otra opción. (Y eso no es difícil, porque un gráfico es realmente una clase especial de curva parametrizada, si hacemos FNa (t) = t. Pero generalmente, uno piensa en ellas de diferente manera).

El usuario va a tener que **estipular** esas funciones. Podríamos tener una línea en el programa que dijera algo así como:

7777 DEF FNb (t) = tonterías o bobadas.

y pedirle al usuario que **editara** esa línea según la función que deseara. Pero ¿no sería mejor permitirle que **impusiera** la función elegida justo en el momento de rular el programa?

El comando VAL (Fácil Programación) viene que ni pintado para esta clase de aplicación. Si el usuario asigna a FNb (t), digamos, una **litérica** f$, luego podremos evaluarla simplemente diciéndole al ordenador VAL f$. Por ejemplo, si t = 71 y f$ = "t * t", vemos que

VAL f$ = VAL "t * t" = 71 * 71 = 5Ø41

que **sí** es el valor de la función "cuadrado" para t = 71.

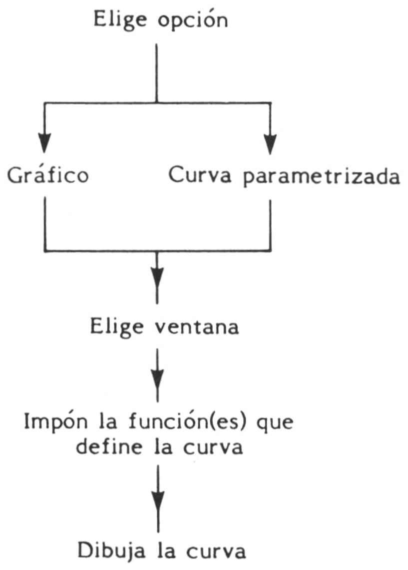
También supondría una tontería si solamente pudiéramos dibujar curvas cuyas coordenadas estuvieran acotadas de Ø a 155 y de Ø a 175 (el problema habitual de desplazar y estrechar ejes, mencionado en Fácil Programación). Una técnica standard es fijar una **ventana** que cubra el área deseada, y transformar las coordenadas adecuadamente durante la rutina que dibuja (figura 16.2).

De modo que un bosquejo de la estructura se muestra en la siguiente página. Dejando los detalles para que se preocupen de ellos mismos (programación **de arriba abajo**; véase Fácil Programación) escribimos el programa principal:

```
100 PRINT 'ESPECTROGRAFO'
110 PRINT ''OPCIONES:''' 1.Curva Parametrizada
                         2.Grafico'
```




```
120 INPUT 'Elige numero de opciones';opcion
130 GO SUB ventana
140 CLS
150 IF opcion=1 THEN  GO SUB parametro
160 IF opcion=2 THEN  GO SUB grafico
170 STOP
```



Bosquejo de estructura para el dibujo de curvas

SUBRUTINAS

Y llegamos aquí con tres subrutinas pendientes de escribir: **ventana, parámetro** y **gráfico.**

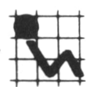
La de **ventana** es la pura simplicidad.

```
 20 LET ventana=500
500 REM ventana
510 INPUT 'Coordenadas ventana:',
    'izquierda','derecha','abajo','arriba',wl,wr,wb,wt
520 RETURN
```

Ya he dicho que **gráfico** es una clase especial de paramétrico, así que, obviamente la cosa a hacer es escribir primeramente paramétrico y luego esperar que **gráfico** encaje fácilmente en ella.

Para pintar una curva parametrizada, con t como parámetro, necesitamos saber dos cosas: la gama de valores de t, y el tamaño de los intervalos dentro de esa gama en que se van a computar los valores de la curva. Por lo tanto, **parámetro** tiene que pedir esos datos, y luego dibujar la curva.

```
  26 LET parametro=2000
2000 REM parametro
2010 INPUT 'Gama de valores:',
     'cota izquierda','cota derecha',tl,tr
2020 INPUT 'Numero de intervalos? ';ns
```



```
2030 INPUT 'Especifica 'x' e 'y'','como funciones de t',x$,y$
2040 LET intervalo=(tr-tl)/ns
2050 GO SUB dibujo
2060 RETURN
```


Figura 16.2 El área de pantalla usada como ventana

Esto nos deja la mayor parte del trabajo para la subrutina **dibuja** que todavía no hemos escrito. Vamos bien! La subrutina **gráfico** funciona de manera muy similar, y hace el "desvío" a **dibujo** en el momento oportuno:

```
  24 LET grafico=1500
1500 REM grafico
1510 LET tl=0: LET tr=255
1520 LET ns=255
1530 INPUT 'Funcion de 't' que deseas';y$: LET x$='t'
1540 LET intervalo=1
1550 GO SUB dibujo
1560 RETURN
```

Deliberadamente la he escrito así para resaltar la analogía con **parámetro. Gráfico** establece todas las mismas variables que **parámetro** excepto para y$, que es la función que tú impondrás, y luego encarga el **dibujo** en, exactamente, la misma forma. (Tú **puedes** sustituir las últimas dos líneas por GO TO 2Ø5Ø, pero el buen estilo en programación sugiere lo contrario, siempre y cuando no desperdicies un montón de memoria o de tiempo).

No podemos seguir retrasando el momento de la verdad ni un momento más...



```
  28 LET dibujo=2500
2500 REM dibujo
2510 LET t=tl
2520 LET u=VAL x$: LET v=VAL y$
2530 GO SUB transforma
2535 LET xo=u: LET yo=v
2540 IF FN o(u,v) THEN  PLOT u,v
2550 FOR t=tl+intervalo TO tr STEP intervalo
2560 LET u=VAL x$: LET v=VAL y$
2570 GO SUB transforma
2575 LET ur=u: LET vr=v
2580 GO SUB testigo
2590 GO SUB d(fl)
2595 LET xo=ur: LET yo=vr
2600 NEXT t
2610 RETURN
```

...Bien, puede que sí podamos después de todo. Nos las hemos arreglado para inventar tres nuevas subrutinas (una de las cuales, encima tiene cuatro partes):

- **transforma** que convierte las variables de manera que la ventana escogida encaje exactamente en el área de pantalla.
- **testigo** que nos indicará si los puntos a unir por **dibujo** están ambos én la pantalla, ambos fuera de ella, o dentro y fuera respectivamente. Y fijará la variable testigo fl a uno de los valores 1, 2, 3, 4, dependiendo de la combinación precisa de posiciones.
- **d (fl)** que son realmente **cuatro** rutinas **d (1)** a **d (4)**, correspondiendo a cada uno de los valores del testigo, porque las acciones que se necesitan son bastante diferentes de un caso a otro.

Hemos metido también una nueva función definida por el usuario FN O (véase Capítulo 3). Se supone que va a ser el testigo de que todo está "in pantalla". Es decir, definimos

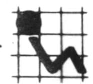
```
1 DEF FN o(x,y)=x>=0 AND x<=255 AND y>=0 AND y<=175
```

por lo que FN O (x, y) = 1 si (x, y) está **dentro** de la pantalla, y FN O (x, y) = Ø si (x, y) está **fuera** de la pantalla. ¿Verdad que es bonito? Desde luego que puedes hacerlo de otras maneras: mi otro yo, probablemente definiría un testigo denominado **inpantalla** y escribiría 254Ø IF inpantalla THEN... De hecho, así funciona con inpantalla = FN O (x, y). Todo lo que sea posible para hacer que el listado se parezca más al Inglés de la Reina y no a la Ley de la Relatividad de Einstein.

Puede que sientas que no vamos a ningún lado todavía, porque no hemos abordado el problema central de **dibujar** realmente. Ten fe: ¿no **sientes** que el problema se está haciendo más pequeño a medida que redondeamos las esquinas y lo troceamos en cachos más pequeños?

Transformar la ventana adecuada es fácil si tienes seis años de entrenamiento matemático, tal como tengo yo:

```
  30 LET transforma=3000
3000 REM transforma
3010 LET u=(u-wl)/(wr-wl)*255:
     LET v=(v-wb)/(wt-wb)*175
3020 RETURN
```



E incluso puedes comprobar que tengo razón. Lo que queremos es que las u-coordenadas wl y wr, se transformen en valores de Ø a 255; y que las wb, wt se transformen de Ø a 175. Ahora, poniendo u = wl en la parte derecha nos da u = Ø en la izquierda; y poniendo u = wr nos da u = (wr - wl) / (wr - wl) * 255 = 1 * 255 = 255... ¡maravillas de la técnica! Y wt, wb funcionan de la misma manera.

Y volviendo al ovillo:

```
  32 LET testigo=3200
3200 REM testigo
3210 LET fl=FN o(xo,yo)+2*FN o(u,v)+1
3220 RETURN
```

Que funciona de la siguiente forma: supongamos que queremos unir el punto **viejo** (xo, yo) al punto **nuevo** (u, v), tendremos:

Punto viejo	Punto nuevo	Valor de fl
dentro	dentro	4
dentro	fuera	2
fuera	dentro	3
fuera	fuera	1

(donde dentro/fuera se refiere a dentro o fuera de la pantalla, no de otro lado). Mediante el valor de fl distinguimos así los posibles casos.

¿Por qué distinguirlos? La acción requerida es:

Valor de fl	Acción requerida
1	Dibujar la parte de línea entre los puntos viejo y nuevo, que cae dentro de la pantalla (si hay algo)
2	Unir el punto viejo al extremo de la pantalla, a lo largo de la línea dirigida al punto nuevo.
3	Unir el borde de la pantalla al punto nuevo, a lo largo de la línea procedente del punto viejo.
4	Unir el punto viejo al punto nuevo.

La razón de la acción (1) es que **puede** suceder que, mientras que ambos puntos, viejo y nuevo caen fuera de la pantalla, parte de la línea entre ellos debiera caer **dentro** de ella, por lo que la dibujamos (véase figura 16.1).

RUTINAS DE DIBUJADO

Finalmente, aquí tenemos todas, escritas en el orden más fácil para mi cerebro en este momento.

```
34 DIM d(4)
37 LET d(1)=3800
38 LET d(2)=3600
39 LET d(3)=3700
40 LET d(4)=3500
```




```
3500 REM d(4)-ambos dentro
3510 DRAW u-PEEK 23677,v-PEEK 23678
3520 RETURN
3600 REM d(2)-viejo dentro,nuevo fuera
3610 LET x1=xo: LET y1=yo: LET x2=u: LET y2=v
3620 GO SUB seg
3630 LET z=1
3640 IF u<>xo AND SGN (u-x(1))<>SGN (x(1)-xo) THEN  LET z=2
3650 IF u =xo AND SGN (v-y(1))<>SGN (y(1)-yo) THEN  LET z=2
3660 DRAW x(z)-PEEK 23677,y(z)-PEEK 23678
3670 RETURN
3700 REM d(3)-viejo fuera,nuevo dentro
3710 LET x1=xo: LET y1=yo: LET x2=u: LET y2=v
3720 GO SUB seg
3730 LET z=1
3740 IF u<>xo AND SGN (u-x(1))<>SGN (x(1)-xo) THEN  LET z=2
3750 IF u =xo AND SGN (v-y(1))<>SGN (y(1)-yo) THEN  LET z=2
3760 PLOT x(z),y(z): DRAW u-x(z),v-y(z)
3770 RETURN
3800 REM d(1)-ambos fuera
3810 LET x1=xo: LET y1=yo: LET x2=u: LET y2=v
3820 GO SUB seg
3830 PLOT x(1),y(1): DRAW x(2)-x(1),y(2)-y(1)
3840 RETURN
```

Todo el uso con el signo es para determinar cuál de los puntos x (1), y (1) o x (2), y (2) es el correcto a emplear. Descártalo si odias las matemáticas.

Eso es casi todo. Las variables xo e yo que son las coordenadas del punto viejo, no han sido dotadas de ningún valor todavía. El sitio correcto es en:

```
2535 LET xo=u: LET yo=v
```

y también ha de ser:

```
2575 LET ur=u: LET vr=v
2595 LET xo=ur: LET yo=vr
```

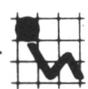
COMPROBACION

Ahora estamos dispuestos a comprobar. Rúlalo y vete tecleando:

Opción	1			
Ventana	-5	5	-5	5
Gama del parámetro	-5	5		
Número de intervalos	1ØØ			
Funciones de t	t	t		

Todo está bien: una línea (no enteramente recta) sube de la parte inferior hacia la parte superior. (¿No lo haces ¡Puede que alguien esté abobado!

¿Algo un poco más difícil? Vamos a por una parábola, parte de la cual está fuera de la pantalla. (La primera comprobación no usó las subrutinas d (1)-d (3) en absoluto... y eso que era el punto principal de todo el ejercicio!).



Opción	1			
Ventana	-5	5	-1	20
Gama del parámetro	-5	5		
Número de intervalos	1ØØ			
Funciones de t	t	t*t		

La figura 16.3 muestra el resultado. Es bonito -pero es una parábola muy castiza...

Puede que te sorprenda o no, saber cuánto tardé en encontrar la pifia. Demasiado tiempo -debía estar bastante cansado. Un rastreo imprimiendo en papel, me mostró que el criminal era el encargado de la gestión de 51) -unir dos puntos, ambos fuera de la pantalla. Pero, ¿qué es lo que está mal hecho?

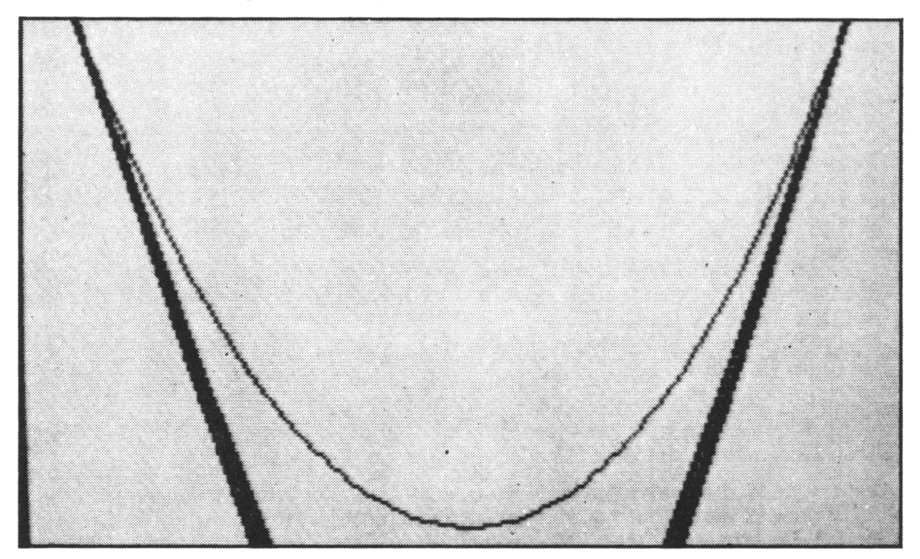


Figura 16.3 Una parábola ! Bueno, casi...

Finalmente, caí en la cuenta de lo que era obvio. Si los dos puntos a unir, están ambos fuera de la pantalla, en el **mismo** lado, la **línea** que los une puede caer dentro de la pantalla aunque el **segmento** que hay entre ellos no caiga (véase figura 16.4)

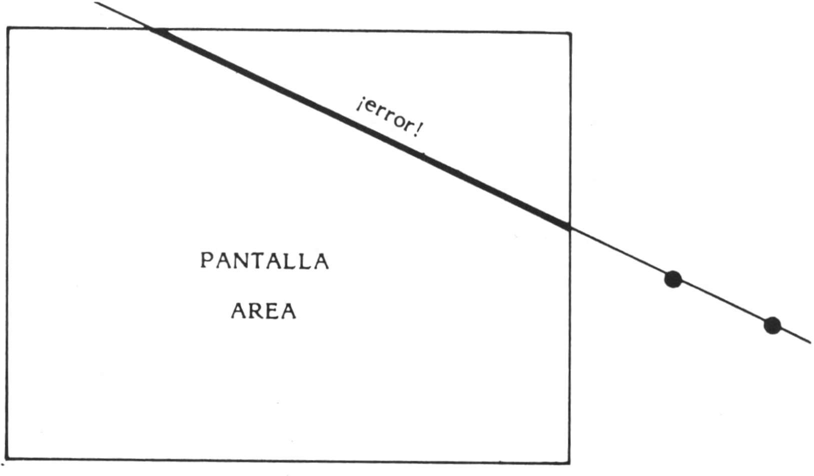


Figura 16.4 El origen de la pifia



Esto se arregla fácilmente, pero usando otra subrutina:

```
  33 LET comprueba=4000
4000 REM comprueba
4010 LET segon=0
4020 IF x1 =x2 AND SGN (y1-y(1))=SGN (y2-y(1)) THEN  LET segon=1
4030 IF x1<>x2 AND SGN (x1-x(1))=SGN (x2-x(1)) THEN  LET segon=1
4040 RETURN
```

Esto estipula otro testigo llamado **segon,** que es 1 cuando se alcanza la situación de la figura 16.4. Para hacer el encargo a esta subrutina, añade:

```
3825 GO SUB comprueba: IF segon THEN  RETURN
```

Al repetir la prueba con la parábola... funcionó! Finalmente, un test mucho más restrictivo:

Opción	1			
Ventana	-.7	.7	-.3	.6
Gama del parámetro	Ø	2.1		
Número de intervalos	3ØØ			
Funciones de t	SIN (11 * PI * t)		COS (13 * PI * t)	

Es una figura de Lissajous (véase Fácil Programación) pero la ventana la he elegido de manera que se sale repetidamente de la pantalla y retorna de nuevo. El resultado aparece en la figura 16.5 y queda bastante claro que el programa hace lo que se supone debe hacer.

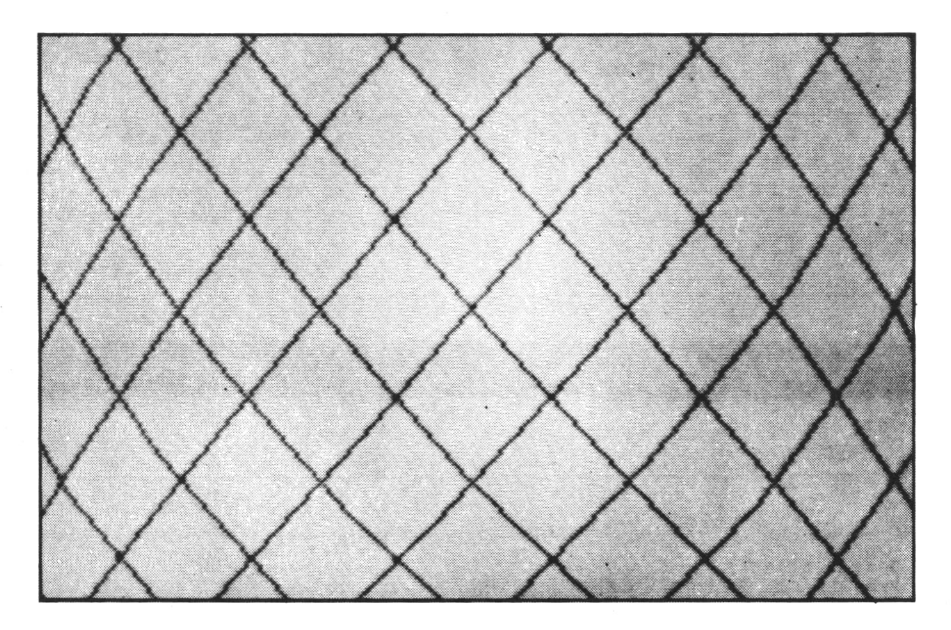


Figura 16.5 Una figura de Lissajous saliendo y entrando de la pantalla muchas veces, hace una prueba excelente.



Proyecto

Este es un programa genuinamente útil. Ensaya con tus propias elecciones de valor para opción, ventana y funciones, empezando con las sugerencias que te mencionamos más adelante. Si tienes dudas, usa las funciones anteriores y simplemente varía las otras opciones una a una. Un buen manantial de ideas es el libro **Un Libro de Curvas** por E.H. Lockwood, Cambridge University Press.

Está claro que todavía son posibles algunas mejoras. En algunos sitios hay trozos de código muy similares que se usan más de una vez -por lo que, sin ninguna duda, una subrutina acortaría el listado. También una opción para volver a pasar el programa cambiando sólo una variable determinada, sería provechoso. Puedes incluso, ser capaz de plantear un enfoque más inteligente a todo el problema. Uno de los peligros de la programación de arriba a abajo, es que si seleccionas para empezar una estrategia mala, tiendes a quedarte pegado a ella.

Puedes también añadir rutinas extra. ¿Quieres dibujar los ejes? ¿Marcar las escalas? ¿Varias curvas superpuestas? Yo dejo esos proyectos a aquellos que se sientan inclinados a hacerlo.

SUGERENCIAS PARA CURVAS

Opción 2: Gráfico

(a)	Catenaria: ventana EXP t + EXP (-t)	-5	5	-1Ø	1ØØ
(b)	Cisoide: ventana t/SQR (1Ø - t)	-5	5	-2	2
(c)	Parábola de Neile: ventana (t * t)↑(1 / 3)	-1Ø	1Ø	-1	1Ø
(d)	Serpentina: ventana 2 * t / (4 + t * t)	-1Ø	1Ø	-1	1
(e)	Estrofoide: ventana t * SQR ((2 - t) / (2 + t))	-1.5	2	-5	1

Opción 1: Curva parametrizada

(f)	Coclioide: ventana	-2Ø	2Ø	-2Ø	2Ø
	gama de parámetros	-3Ø	3Ø		
	número de intervalos	5ØØ			
	t * SIN t * COS t				
	t * SIN t * SIN t				



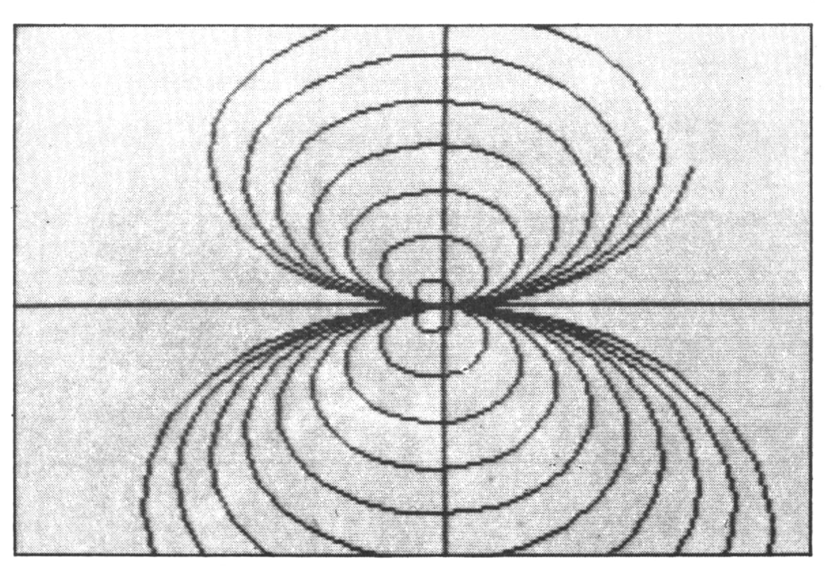


Figura 16.6 La coclioide

(g)	Caracola:	ventana	-1Ø	1Ø	-1Ø	1Ø
		gama	-PI	PI		
		intervalos	1ØØ			
	(3 + 5 * COS t) * COS t					
	(3 + 5 * COS t) * SIN t					
(h)	Roseta:	ventana	-1.2	1.2	-1.2	1.2
		gama	Ø	PI		
		intervalos	5ØØ			
	COS (11 * t) * COS t					
	COS (11 * t) * SIN t					

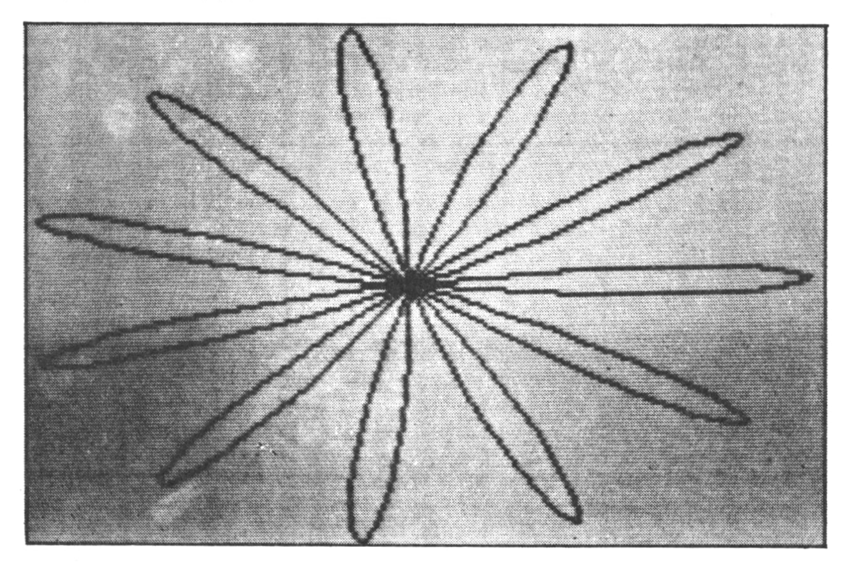


Figura 16.7 Una roseta de 11 pétalos



(i)	Pseudo-Lissajous:	ventana	-1Ø	1Ø	-1Ø	1Ø
		gama	-5	5		
		intervalos	5ØØ			

```
1Ø / (1 + t * t) * SIN (3 * t)
1Ø / (1 + t * t) * SIN (5 * t)
```

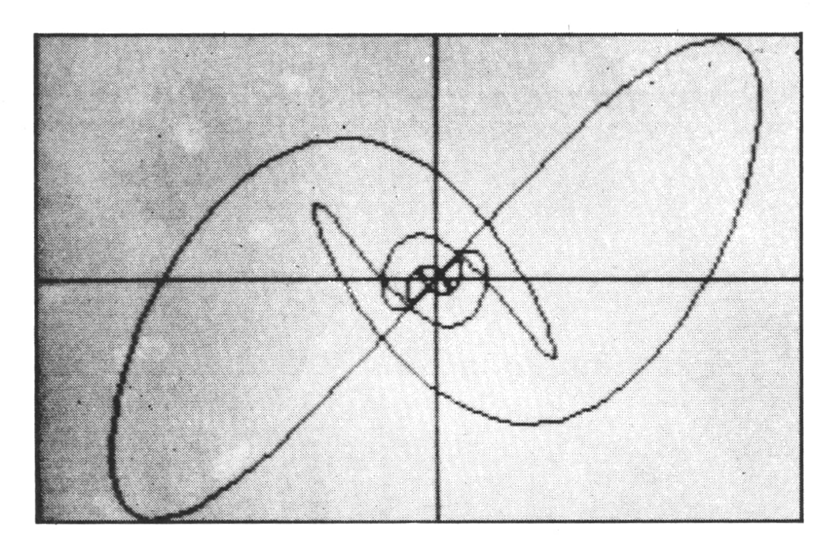


Figura 16.8 Una elegante figura pseudo-Lissajous

La mayoría de los ficheros tienen mucho en común. ¿Por qué no tratarlos todos de igual manera?

17 Sistemas de Gestión de Archivos

Avancemos en el desarrollo de las ideas sobre ficheros (capítulo 9). Allí, se supuso que antes de que puedan escribirse programas de creación, mantenimiento y escrutinio, tenemos que saber los rasgos exactos del fichero con el que estamos tratanto. Y sin embargo, todos los ficheros se van a parecer muchísimo. Simplemente variará el número de campos por registro y sus longitudes. De forma que no sería difícil modificar **inreg**, por ejemplo; cambiando todas esas constantes (30, 36, etc.) por variables. Como un eminente científico irlandés en computadoras señaló: "Todas tus constantes debieran ser variables".

Más despacio, sin embargo. ¿Cómo sabrá **inreg** cuáles son las longitudes de los campos en un caso particular? Bien, ¿por qué no permitir que la rutina de inicialización las pregunte al usuario cuándo se está creando el fichero, o que las tome del bloque de cabecera en caso contrario? La forma del bloque de cabecera se está complicando un poquito más ahora, así que observémoslo con algún detalle.

EL BLOQUE DE CABECERA

Básicamente, va a haber dos tablas. Una contiene un nombre suministrado por el usuario para cada campo (así que es una tabla de literales), y la otra guardará el número de octetos reservados para cada campo. Ya no necesitamos saber el número de octetos por registro, porque es la suma de todas las longitudes de los campos, pero sí necesitamos todavía el número de registros por bloque, de manera que los incorporemos dentro de una tabla numeral. Hay otra consideración más. Los campos pueden contener información numérica o litérica, y probablemente necesitaremos manejarlos de forma diferente en cada caso. Así que tendría sentido asegurar que los campos se distinguen de esa forma cuando se estipula. Podríamos tener otra tabla con esta información, pero yo voy a rotular cada nombre de campo con una "l" o "n" en el primer octeto, para indicar si es de índole literal o numeral. Así que nuestras tablas, tendrán la apariencia dada en la figura 17.1.

En ella, supongo que el fichero va a registrar las transacciones de cuentas bancarias. Tenemos una fecha con seis caracteres (y es de índole literal porque no vamos a hacer ningún cálculo aritmético con ella, si lo quisieramos, sería mejor tener 3 campos numéricos separados, ndía, nmes y naño); el número del cheque; la cantidad de dinero (ambos numerales) y un campo descriptivo (literal) en el que puedan reseñarse hasta 17 caracteres para describir la transacción. Los números de cheque tienen siempre 6 dígitos (los míos por lo menos), y 8 dígitos en el campo del importe que permiten reseñar hasta 99.999,99 ptas., que desde luego es una cifra optimista para **mi** cuenta del banco. (Observa que se usa una posición para la coma decimal). El número de registros por bloque es 2Ø y, para que los nombres de los campos concuerden con sus longitudes, n$ (1) se deja en blanco. A mí me hace siempre feliz, que exista ese espacio extra y extraño, porque siempre me deja un escape si se me ha olvidado algo. Dado que ambas tablas tienen 11 elementos, hay un sitio para 1Ø campos por registro. Desde luego, no presenta ningún problema alterar esto si crees que es un poco restrictivo.



	n$		w	
sin uso →		1	20	← registros por bloque
	fecha	2	6	
	ncheque	3	6	
	importe	4	8	
	descrip	5	25	
		6		
		7		
		8		
		9		
		10		
		11		
	← 10 bytes de ancho →			

Figura 17.1 Trazado del bloque de cabecera

PREPARANDO UN FICHERO

La gestión de **iniciar** quedará como sigue:

```
9500 DIM n$(11,10): DIM w(11): LET ip=0: LET op=0:
     LET inbc=0: LET exbc=0: LET nrb=0
9505 INPUT "Nombre fichero de entrada:";f$
9510 IF f$="nulo" THEN  GO SUB preparar: GO TO 9525
9514 PRINT "Marchando un pinchado de cinta"
9515 LOAD f$+"h1" DATA n$()
9521 PRINT "Amputando acabose la RASCAZON"
9525 INPUT "Nombre del fichero de salida:";g$
9530 FOR p=2 TO 11
9535 LET nrb=nrb+w(p)
9540 NEXT p
9545 DIM i$(w(1),nrb): DIM o$(w(1),nrb)
9547 IF g$="nulo" THEN  RETURN
9550 SAVE g$+"h1" DATA n$()
9555 SAVE g$+"h2" DATA w()
9557 PRINT "Amputandose acabose la GRABAZON"
9560 RETURN
```




Es casi lo mismo que antes, excepto que si tecleas "nulo" como nombre del fichero de entrada, reclamará **preparar** que generará una descripción de un fichero nuevo. Si no, carga la descripción del fichero a partir de los dos primeros bloques del fichero de entrada (que, para un fichero llamado "pepe", será "pepeh1" y "pepeh2"). Calcula el número de octetos por registro (líneas 9530 a 9540) y prepara los buzones de entrada y salida a partir de esa información. Finalmente, si el fichero de salida no es "nulo", escribe sus bloques de cabecera.

Redactaremos la subrutina **preparar** a partir de la 9400:

```
9400 INPUT "Numero de campos:";nf
9405 CLS
9410 FOR p=2 TO nf+1
9415 PRINT AT 10,2;"Campo ";p-1
9420 INPUT "Nombre del campo:";n$(p)
9425 INPUT "Numero de octetos:";w(p)
9430 NEXT p
9435 CLS
9440 INPUT "No. de registros por bloque:";w(1)
9445 RETURN
```

No hay mucho que comentar aquí. La gestión **preparar** ejecuta simplemente un buble FOR, una vez para cada campo, tecleando el nombre del campo y la longitud de los elementos apropiados. Hay una pequeña cuerda de violín a tocar aquí, y es el hecho de que el campo 1 realmente ocupa el elemento segundo de la tabla. Finalmente, se especifica el número de registros por bloque y se impone en w (1).

Ahora vamos derechos a la base. Podemos volver a escribir **inreg** usando la versión de formato prefijado para que nos dé pistas de cómo abordar esta rutina de propósito más general.

La vieja comenzaba con:

```
8000 DIM a$(336)
```

siendo 336 la longitud de un registro.

Ahora **iniciar** ha evaluado eso y lo ha puesto en nrb. Así que tenemos:

```
8000 DIM a$(nrb)
```

La siguiente línea fue:

```
8010 INPUT "Artista ";a$( TO 30)
```

Así que, en términos más generales, lo que nos gustaría hacer es tener un bucle en que la línea equivalente dijera algo así como:

INPUT "otro campo"; a$ (comienzo de campo TO fin de campo)

y que eso se repitiera para cada campo. No puedes escribir algo como:

```
10 LET p$="otro valor"
20 INPUT p$;nv
```

porque, aunque es perfectamente válido en BASIC, su significado es diferente de lo que queremos. La línea 20 no nos dice que "presente lo que haya en p$ como si fuera un mensaje y luego impusiera el valor tecleado en nv". Dice que "imponga la ristra que tecleemos en p$, y luego el valor que tecleemos en nv". Sin embargo, **si pones paréntesis a p$,** se produce el efecto deseado.



IMAGEN EN PANTALLA

Mientras este truco es simple y conveniente, puede confundir un poco al usuario, él solamente ve un campo de un registro en cada momento, y sería de más ayuda si viera **todo** el registro que se está formando. Además, él no tiene ninguna indicación sobre la longitud del campo que puede usar.

La solución simple es formar el registro en la pantalla usando exposición de mensajes para cada campo y copiando cada valor impuesto para que concuerde con la pregunta apropiada. Será una técnica familiar para tí si te salieron los dientes con el ZX81, ¡que no permitía mensajes de preguntas en las instrucciones INPUT!

Así que lo que nos gustaría, es que en la pantalla se mostrara algo como este ejemplo de cuenta bancaria:

l:	fecha	Ø6Ø482
n:	nº de cheque	317462
n:	cantidad	-71.37
l:	descripción	Silla de oficinas

donde las partidas subrayadas son valores impuestos por teclado, que inmediatamente vuelven a presentarse en pantalla. El subrayado realmente aparecerá, y mostrará la anchura máxima de cada campo.

Observa varias cosas: primeramente, el tipo de campo se ha separado del nombre y se presenta simplemente como recordatorio para el usuario. En segundo lugar, el importe se muestra como negativo, y verás por qué es útil ese convenio posteriormente. En tercer lugar, el usuario hace una observación en el campo de descripción de que el material de oficina es deducible de impuestos. No está usando el sistema muy sensatamente, porque es probable que desee una lista de partidas deducibles al final del año, y eso significa examinar parte de un campo. Debiera haberlo pensado antes de comenzar y usar un quinto campo para identificar las partidas deducibles.



En todo caso, volviendo al problema:

`8010 CLS : LET comienzo=1`	borra la pantalla de forma que la imagen del registro no se vea amontonada, establece el puntero para el comienzo de a$...
`8020 FOR p=2 TO 11`	y se mete en un bucle
`8025 IF n$(p,1)="□" THEN GO TO 8120`	comprueba si es el último campo
`8030 PRINT AT p,0;n$(p,1);":";AT p,4;n$(p)(2 TO )` `8040 FOR c=1 TO w(p)` `8050 PRINT AT p,14+c;"_"` `8060 NEXT c`	presenta la carátula o plantilla para el campo



Como a$ es una variable litérica, necesitamos saber dónde comienza y termina la cadena que corresponde a este campo:

```
8070 LET termino=comienzo+w(p)-1
```

En la primera ronda, comienzo = 1, porque así fue asignado en la línea 8010 y término = 6, lo que significa que podemos escribir:

```
8080 INPUT (n$(p)(2 TO ));a$(comienzo TO termino)
```

y el efecto será preguntar con la palabra "fecha" y transferir eso a a$ (1 TO 6). Ahora, ponemos eso mismo sobre la pantalla:

```
8090 PRINT AT p,15;a$(comienzo TO termino)
```

Finalmente, debemos fijar la nueva posición de "comienzo":

```
8100 LET comienzo=termino+1
```

y cerrar el bucle:

```
8110 NEXT p
```

Pasar el resultado a r$ y salirnos de la subrutina:

```
8120 LET r$=a$
8130 RETURN
```

COMPROBANDO

En este momento, podemos escribir un par de rutinas de comprobación para ver que todo funciona. Primeramente necesitamos crear un fichero. La rutina que usamos para crear la colección de discos, funcionará sin modificación, simplemente para recordarte:

```
100 GO SUB iniciar
110 GO SUB inreg
120 GO SUB punza
130 INPUT "Alguno mas?(s/n)";q$
140 IF q$="s" THEN  GO TO 110
150 GO SUB cierre
160 STOP
```

Cuando rulas este programa, **iniciar** preguntará el nombre de fichero de entrada, a lo que responderás "nulo", desde luego, y se te solicitará la descripción del registro. Pudieras usar la cuenta bancaria 1 como un ejemplo suficientemente simple. Haz el tamaño de bloque pequeño, digamos 5, de manera que no tengas que teclear demasiados registros antes de que se active el mecanismo de agrupamiento en bloques. De esa manera, todo se comprueba sin aporrear demasiado el teclado. Finalmente, impón 10 o 15 registros, para formar el fichero de prueba.

Ahora necesitamos saber si la información se ha guardado correctamente. Sustituye las líneas 110-160 con:

```
110 GO SUB pinza
120 IF r$( TO 2)="}}" THEN  STOP
130 PRINT r$
140 GO TO 110
```



Cuando pases este programa, sacarás registros como:

```
1208821234592 1.76□□□ pipas
```

si has empleado el formato de registro del ejemplo de la cuenta bancaria.

Ahora bien, sabemos que los primeros 6 octetos representan una fecha, y eso es 12/Ø8/82; que viene detrás el número de cheque (123459) y un importe (21.76 -observa los 3 blancos, debido a que hay sitio para 8 octetos en la definición del registro) y finalmente, una descripción. Pero salidas como esa, no son cómodas para el usuario. Así que lo que realmente necesitamos es una rutina que desentrañe r$ en campos separados. Llamémosla **exreg** dado que realiza la función opuesta a **inreg**,y coloquémosla a partir de 82ØØ:

```
8200 LET comienzo=1
8210 FOR p=2 TO 11
8220 IF n$(p)="□" THEN  PRINT : RETURN
8230 PRINT n$(p,1);":";TAB 4;n$(p)(2 TO );
8240 LET termino=comienzo+w(p)-1
8250 PRINT TAB 15;r$(comienzo TO termino)
8260 LET comienzo = termino + 1
8270 NEXT p
8280 PRINT : RETURN
```

No es una sorpresa que esta rutina tenga un parecido más que pasable a **inreg**, pero hay una diferencia importante. Queremos que los registros se "desrrollen" continuamente, en lugar de que aparezcan en la misma posición de la pantalla. Si lo arreglaramos para esto último, tendrías que ser un lector extrarápido para ver algo! Por lo tanto, no es bueno "PRINT AT". Para conseguir la tabulación horizontal, previamente provista por las coordenadas en una instrucción "PRINT AT", uso la función "TAB". Si no la has usado anteriormente, puedes pensar que es equivalente a "PRINT AT" sin especificar el renglón. En otras palabras, si dices

PRINT AT 2, 15;...

estás especificando el renglón 2.

PRINT TAB 15;...

da la misma posición de columna, pero en el siguiente renglón disponible, **donde quiera** que esté.

Ahora, todo lo que tenemos que hacer es cambiar la línea 13Ø para que sea:

13Ø GO SUB exreg

y, cuando se rula el programa resultante, cada registro aparece en forma legible sobre la pantalla. Desde luego, es poco probable que desees un listado completo del fichero en la pantalla. Sería más sensato sacarlo por impresora. Así que tendríamos una rutina llamada **prexreg** que enviaría un registro a la impresora. Sería idéntica a **exreg**, excepto que todas las PRINT serían cambiadas a LPRINT.

EL DISEÑO GENERAL

Ahora, ya disponemos de todas las herramientas necesarias, para construir propiamente nuestro sistema de gestión de datos. Así que tomémonos un descanso de la codificación, retrocedamos unos pocos pasos, y consideremos el diseño general.



En el primitivo sistema de manejo de ficheros, llevamos a cabo tres rutinas: crear, mantener y escrutar. Necesitamos todas esas, más unas cuantas extra. En esta ocasión las armonizaremos mediante un menú en el mismo programa principal. Por tanto no tiene mucho objeto tener "mantener", como una de las opciones, y luego preguntar inmediatamente si es para "agregar" o "suprimir" registros. Pudieramos también tener "agregar" y "suprimir" como opciones primordiales.

Nuestro programa principal se parecería entonces a éste:

```
 10 CLS
 20 PRINT AT 0,5;"Archivero Spectrum": GO SUB iniciar
 26 CLS
 30 PRINT AT 2,0;"Las opciones son:"
 40 PRINT AT 3,11;"1/ crear"
 41 PRINT AT 4,11;"2/ agregar"
 42 PRINT AT 5,11;"3/ suprimir"
 43 PRINT AT 6,11;"4/ escrutar"
100 INPUT "Teclee numero de opcion:";opt
110 GO SUB 200*opt
120 GO TO 26
```

¿Estoy oyendo murmullos de descontento? ¿Hay alguien por ahí musitando que hace un minuto decía que necesitaba implementar algunas rutinas extras, y al siguiente minuto sólo permite aquéllas que ya habíamos planteado? Tienes toda la razón, muchacho. Pero observa cómo me lo he montado para abordar cualquier otra rutina. Simplemente añades otra línea a las opciones:

```
 44 PRINT AT 7,11;"5/ lo que se te antoje"
```

y luego colocas la subrutina correspondiente a partir de la línea 1000, dado que el mecanismo que guía el programa principal hacia la subrutina correcta, simplemente multiplica el número de la opción por 200.

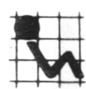
Así que	crear	está en la 200
	agregar	está en la 400
	suprimir	está en la 600
	escrutar	está en la 800
	lo que se te antoje	está en la 1000

y así sucesivamente.

Además, este es un mejor enfoque para construir el sistema en forma confeccionada y estática, que no permita ninguna ampliación. Sólo después de haber usado el sistema durante bastante tiempo, es cuando se empieza a dar cuenta de sus limitaciones y se desea implementar una rutina que domine el mundo o haga lo que sea. Haciendo las cosas de esta manera, puedes modificar el programa en el momento en que la inspiración (o, con más probabilidad, frustración) te sobrevenga.

INICIALIZACION

Una cosa más a considerar: **iniciar** es citada antes de que se ofrezca ninguna de las opciones. Ahora insertarlo en cada rutina, y responde ante ella más de una vez si quieres hacer varias cosas con los mismos ficheros. Significa, desde luego, que el programa debe volverse a rular para elegir ficheros diferentes. Desafortunadamente, significa también, volver a pensar un poquito sobre la codificación, porque **iniciar** hace algunas cosas más que meramente definir nombres de ficheros.



También fija los punteros y los contadores de bloques, y esas cosas **tienen** que estar asignadas al comienzo de la lectura de un segundo fichero. Así que meteremos una subrutina más dentro del sistema de ficheros en cassette, denominada **arredro** que simplemente "vuelve atrás" esos punteros:

```
9970 LET pin=0: LET pex=0: LET inbc=0: LET exbc=0
9980 RETURN
```

Con esto, la primera línea de **iniciar** puede convertirse en:

```
9500 DIM n$(11,10): DIM w(11): LET lpr=0: GO SUB arredro
```

y desde luego, la línea 1 tiene añadido:

```
LET arredro=9970
```

Finalmente, podemos citar **arredro** en el programa principal antes de volver del bucle hacia el menú:

```
120 GO SUB arredro
130 GO TO 26
```

LAS RUTINAS

La de **crear** ya la tenemos, excepto que necesita renumerarse y una instrucción de vuelta en lugar de una de paro:

```
200 GO SUB inreg
210 GO SUB punza
220 INPUT "Alguna mas?(s/n)";q$
230 IF q$="s" THEN  GO TO 200
240 GO SUB cierre
250 RETURN
```

la de **agregar** es un poco más problemática. Por un lado, no fue previa y separadamente implementada, y por el otro, ya he comentado que es de naturaleza algo primitiva. Así que, aprovecharemos la oportunidad de volver a considerar el problema. Necesitamos cargar todos los añadidos al fichero en una tabla para empezar, y luego, a medida que se lee el fichero, comparar cada uno de ellos con el registro "vigente" para decidir si hacer la inserción o no. Necesitamos también marcar cada posible inserción para indicar si ya ha sido insertado o no. Por el momento, vamos a hacer como el avestruz ante ese problema. Así que:

```
400 INPUT "Cuantos registros?";nr     ]-prepara la tabla
410 DIM b$(nr,lpr)                    ]

420 FOR q=1 TO nr                     ]
430 GO SUB inreg                      ]-carga registros
440 LET b$(q)=r$                      ]  adicionales en b$
450 NEXT q                            ]
```

El siguiente párrafo puede sonar a truco. Tenemos que identificar la parte del registro que vamos a usar como **clave.** En el fichero de cuentas bancarias por ejemplo, podríamos ordenar los registros por fechas o por número de cheque o por cantidad; y como es usual, queremos darle al usuario tanta flexibilidad como sea posible. Así que, citaremos a una subrutina, de cuyos detalles nos ocuparemos posteriormente, que llamamos "sacaclaves". Por el momento, diremos simplemente la gestión que nos gustaría que nos hiciera **sacaclave,** requerirá al usuario el nombre del campo clave y entregará a la rutina que la citó tres valores:

comienzo: el octeto de r$ o b$ (p) al comienzo del campo clave



término:	el octeto de r$ o b$ (p) al término del campo clave
ltipo:	será Ø si la clave es numérica y 1 si es litérica

Definiendo **sacaclave** como una subrutina, tiene la ventaja habitual de parecer que hacemos muy poco trabajo, ya que definimos lo que realiza y podemos usarla antes de preocuparnos realmente de desarrollar cómo lo hace. Sin embargo, anticipandonos un poco, podemos detectar que hay presente un rasgo tipo para una segunda subrutina: ¡porque **suprimir** y **escrutar** también van a necesitarlas!

En todo caso, por el momento, supondremos que ya disponemos de **sacaclave** y vamos a ver cómo la usamos:

```
460 GO SUB sacaclave
```

ORDENANDO

A continuación, tendremos que poner en orden los registros adicionales. No podemos hacerlo antes porque no sabemos según qué clave ordenar, hasta que **sacaclave** haya sido puesta a trabajar:

```
465 GO SUB ordenar
```

y ahora sí podemos fijar un puntero hacia el primer registro en b$ que va a insertarse en el fichero:

```
470 LET ap=1
```

A partir de aquí, las cosas son bastante directas. Todo lo que necesitamos es comparar b$ (ap) con el siguiente registro del fichero (r$). Si r$ tiene una clave inferior, lo sacamos; si no sacamos b$ (ap). Hay, sin embargo, un punto a vigilar. Si se saca r$ necesitamos coger el siguiente registro del fichero; pero si el que sacamos es un elemento de b$, simplemente tenemos que bombear el valor de ap en la unidad. En ambos casos, debemos ser precavidos contra la lectura más allá del fichero, o más allá del final de la tabla. ¿Qué sucede cuando el fichero de entrada o la tabla a añadir se vacía? Desafortunadamente, cosas diferentes. Así que citaremos a una subrutina **cabosueltos** que nos atará bien atadas estas cosas.

```
480 GO SUB pinza
490 IF ap>nr OR r$( TO 2)="}}" THEN  GO SUB cabosueltos: RETURN
```

Ahora queremos comparar las claves de r$ y b$ (ap). Vamos a sacarlas primeramente y ponerlas en s$ y t$ respectivamente:

```
500 LET s$=r$(comienzo TO termino):
    LET t$=b$(p)(comienzo TO termino)
```

Comparemos ahora s$ y t$. Pero hay un problema. s$ y t$ pueden realmente tener índole numérica o litérica. Si realmente son numéricas, están vistas como litéricas, por lo que entonces, no se consideraría que 123 es idéntico que 123.Ø. Peor todavía, 5 resulta que es mayor que 12!

Así que necesitamos dos subrutinas más, **companum** y **compalit**, que hagan comparaciones numéricas y litéricas respectivamente. Ambas entregarán un valor llamado comp que destaque el resultado de la comparación en la forma siguiente:

Valor de comp	Significado
-1	s$ < t$
Ø	s$ = t$
1	s$ > t$



Puede que barruntes por qué he especificado "=" y ">" cuando todo lo que realmente necesitamos es "<". La razón es que mantengo el ojo sobre las rutinas **suprimir, escrutar** y **ordenar** que también usarán comparaciones, y probablemente ne maneras diferentes a ésta. Así que, es mejor hacer las rutinas tan generales como sea posible.

Ahora podemos escribir:

```
510 IF ltipo THEN  GO SUB compalit
520 IF NOT ltipo THEN  GO SUB companum
```

Puede que no hayas visto este tipo de construcción anteriormente (a no ser que ya hayas leído el capítulo 3). No parece que haya una comprobación detrás de la palabra "IF". La verdad es que toda condición dentro de una instrucción "IF" se evalúa a Ø o a 1, dependiendo de si es falsa o cierta. Por tanto, si escribes:

```
75 IF a=b THEN ......
```

la "a = b" se sustituye por 1 si sus valores son el mismo, y por cero si no lo són. Dado que ltipo sólo posee el valor Ø o el valor 1, estamos ahorrando un paso en la evaluación y el resultado queda más bonito que si pusieramos "IF ltipo = Ø THEN GO SUB companum.

Vamos ahora a comprobar comp. Si comp es menor que cero, queremos sacar r$ y coger el siguiente registro del fichero para sustituirlo:

```
530 IF comp<0 THEN  GO SUB punza: GO TO 480
```

Si no lo es, tenemos que sacar el registro corriente adicional. Pero no olvides que sólo podemos sacar lo que hay en r$, y su contenido corriente tendrá que ser depositado y recuperado antes y después de este proceso:

```
540 LET t$=r$: LET r$=b$(ap): GO SUB punza:
    LET r$=t$: LET ap=ap+1: GO TO 490
```

Observa el irse a la 49Ø al final. No regresamos a la 48Ø porque no hemos vaciado todavía el registro corriente del fichero, así que no necesitamos ningún otro!

MAS SUBRUTINAS

Ahora viene la recapitulación. Las subrutinas citadas con abandono temerario a partir de la rutina de agregación, tendrán que escribirse. Le hemos dado los números de línea en que van a comenzar, y desde luego, tendrán que ser inicializadas al comienzo del programa:

sacaclave	78ØØ
cabosueltos	76ØØ
compalit	74ØØ
companum	72ØØ
ordena	7ØØØ

la de **sacaclave** tiene primero que preguntar al usuario cuál es el campo de clave:

```
7800 DIM k$(9): INPUT "Teclee nombre del campo clave ";k$
```

k$ será un nombre de campo a partir del segundo octeto. En otras palabras, no incluirá el carácter que marcha la índole del campo. Parece más natural teclear "cantidad" y no "ncantidad", ya que la "n" no es esencial porque tenemos esa información dentro de n$.

Ahora tenemos que examinar la tabla n$ buscando k$; pero tendremos que guardar registro de cuántos octetos llevamos examinados. Así que adoptemos este procedimiento: supongamos que es el primer campo el que vamos a examinar, así que comienzo = 1. Si no lo es, daremos un bote a comienzo de w (2), de forma que apunte ahora al comienzo del segundo campo. Si ese no es el que queremos, bombearemos comienzo con w (3) y así sucesivamente.

```
7810 LET comienzo=1
7820 FOR p=2 TO 11
7830 IF n$(p)(2 TO )=k$ THEN  GO TO 7860
7840 LET comienzo=comienzo+w(p)
7850 NEXT p
7860 LET termino=comienzo+w(p)-1
```

(nota: k$ debe tener 9 octetos de longitud porque está siendo comparado con los últimos 9 octetos de cada uno de los elementos de n$)

Ahora podemos identificar la índole del campo examinando el primer octeto del elemento de n$ al que está apuntando p:

```
7870 IF n$(p,1)="c" THEN  LET ltipo=1
7880 IF n$(p,1)="n" THEN  LET ltipo=0
7890 RETURN
```

Realmente, no hizo mucho daño, ¿o sí?

Ahora vamos con **cabosueltos.** Si hemos llegado primero al final del fichero, tenemos que vaciar el buche de registros a insertar. En caso contrario, tenemos que copiar el resto del fichero. Así que la rutina es bastante directa:

```
7600 IF ap>nr THEN  GO SUB copiresto: RETURN
7610 FOR p=ap TO nr
7620 LET r$=b$(p)
7630 GO SUB punza
7640 NEXT p
7650 GO SUB cierre
7660 RETURN
```

y si **copiresto** está en la 77ØØ:

```
7700 GO SUB punza
7710 GO SUB pinza
7720 IF r$( TO 2)="}} " THEN  GO SUB cierre: RETURN
7730 GO TO 7700
```

compalit y **companum** son bastante fáciles:

```
7400 IF s$<t$ THEN  LET comp=-1
7410 IF s$=t$ THEN  LET comp=0
7420 IF s$>t$ THEN  LET comp=1
7430 RETURN

7200 IF VAL s$<VAL t$ THEN  LET comp=-1
7210 IF VAL s$=VAL t$ THEN  LET comp=0
7220 IF VAL s$>VAL t$ THEN  LET comp=1
7230 RETURN
```

LA RUTINA DE ORDENAMIENTO

Finalmente, necesitamos **ordena.** Si has leido **Fácil Programación,** recordarás que presenté un algoritmo de ordenamiento por "burbuja" en los capítulos de depuración. Aquí lo usaremos de nuevo. Y no es la forma de ordenar más eficaz o elegante que se haya imaginado nunca (de hecho, es casi lo contrario) pero **sí es** simple y dado que b$ no es una tabla enorme, no tardará mucho en ejecutarse.



```
7000 LET inc=1
7005 LET testigo=0
7010 FOR p=1 TO nr-inc
7020 LET s$=b$(p)(comienzo TO termino):
     LET t$=b$(p+1)(comienzo TO termino)
7030 IF ltipo THEN  GO SUB compalit
7040 IF NOT ltipo THEN  GO SUB companum
7050 IF comp>0 THEN  LET t$=b$(p): LET b$(p)=b$(p+1):
     LET b$(p+1)=t$: LET testigo=1
7060 NEXT p
7070 IF testigo>0 THEN  LET inc=inc+1: GO TO 7005
7080 RETURN
```

Así, en retrospectiva, la de **agregar** ha requerido bastante más esfuerzo. Pero es probablemente la que tiene más trucos de todas las rutinas del menú.

SUPRIMIR

Ahora iremos cuesta abajo durante un rato para escribir la de **suprimir:**

```
600 INPUT "Cuantos registros?";nr            (halla los límites de los campos
610 GO SUB sacaclave                          clave y los usa para establecer
620 DIM d$(nr,termino-comienzo+1)             la dimensión de la tabla adecuada)

630 FOR p=1 TO nr                           ]
640 INPUT "Clave a suprimir                 ]  coloca la tabla de claves
    (solo clave):";d$(p)                    ]  a suprimir dentro de d$
650 NEXT p                                  ]

660 GO SUB pinza
670 IF r$( TO 2)="||" THEN  GO SUB          ]
    cierre: RETURN                          ]  busca la concordancia entre
680 FOR p=1 TO nr                           ]  la clave de r$ y lo reseñado
690 IF r$(comienzo TO termino)=d$(p)        ]  en d$. Si encuentra una, coge
    THEN  GO TO 660                         ]  el siguiente registro.
700 NEXT p                                  ]  No la ha encontrado, así
710 GO SUB punza                            ]  que saca el registro...
720 GO TO 660                               ]  y consigue otra.
```

Progresando, siempre progresando...

ESCRUTANDO

Antes de precipitarnos donde ni los ángeles se atreven, debiéramos hacer algunas consideraciones serias sobre cómo queremos que se comporte la rutina de **escruta.**
La cosa más simple a hacer, sería permitir que el usuario tecleara una única clave, y luego ir leyendo a través del fichero, mostrando en la pantalla cada uno de los elementos que tengan esa clave. Para decidir si eso sería adecuado, intenta ponerte en la posición de usuario e imaginar la clase de preguntas que le gustaría hacer. Volvamos al fichero de cuentas bancarias como un ejemplo conveniente. Si hay un campo que indica que una partida es deducible de impuestos, entonces el usuario pudiera muy bien desear mostrar todos los registros en que en ese campo dijera "sí". Por otro lado, él podría querer mostrar todos los registros que hacen referencia a cheques sacados por más de 40.000 ptas., o a todos los números de cheque mayores de 318472, o a cheques entre 8 de julio de 1981 y 5 de septiembre de 1982.

En otras palabras, es muy probable que desees considerar una **gama** de teclas y no solamente una, así que debieramos permitirle ambas posibilidades. En segundo lugar, ¿querrá **siempre** mostrar los registros que encuentre durante la búsqueda? Ciertamente, como mínimo le gustará tenerlos impresos. Pero hay otra posibilidad que mejora impresionantemente el aprovechamiento del sistema sin que implique ningún sudor significativo (bueno, no sudor extraordinario) por nuestra parte, es permitirle que los registros encontrados durante la búsqueda sean grabados en un nuevo fichero. De esta manera, se pueden usar búsquedas sucesivas para aislar combinaciones de condiciones. Por ejemplo, si necesitamos una lista de todas las partidas deducibles de impuesto que superen las 20.000 ptas. en cantidad, generaríamos primero un nuevo fichero con todas las partidas deducibles, y luego lo escrutaríamos para segregar todas las partidas de más de 20.000.

Dije que era simple de preparar. Todo lo que entraña, es que habiendo encontrado un registro de los buscados, citemos a la rutina **exreg** para que lo presente en pantalla, **prexreg** para que lo imprima, o **punza** para que lo saque a un fichero.

Así que vamos a por ello:

```
800 GO SUB sacaclave
810 IF ltipo THEN  DIM k$(termino-comienzo+1):
811 INPUT "Clave de busqueda:";k$
820 IF NOT ltipo THEN
    INPUT "Gama de valores__cota inferior:";cotainf,
          "cota superior:";cotasup
```

Realmente se necesita ya un poquito de explicación. Obviamente, la primera labor es determinar qué campo se va a usar como clave durante el recuento del fichero, de aquí que citemos a **sacaclave.** Luego, si la clave elegida es de índole litérica, el concepto de gama apenas tiene significado (a no ser que quieras tratar con una gama de teclas alfabéticas, como todos los nombres entre García y Pérez, pero aquí no estamos considerando esto. Es suficientemente fácil si necesitas hacerlo. Así que únicamente solicitamos una única clave de búsqueda (que debe concordar con la longitud del campo correspondiente del registro -de ahí que usemos DIM). Sin embargo, si la clave es de índole numérica, sí preguntamos por una gama de claves.

Luego, necesitamos saber qué opción de salida se va a elegir:

```
830 INPUT "Pantalla(1),Impresora(2)Cassette(3):";opt
```

Ahora, ya podemos comenzar el escrutinio del fichero:

```
840 GO SUB pinza
850 IF r$( TO 2)="⦀ " AND opt=3 THEN  GO SUB cierre: RETURN
860 IF r$( TO 2)="⦀ " THEN  INPUT "Pulse para continuar";k$: RETURN
```

Esto es un trozo de código ligeramente embrollado. El problema es que, cuando se identifique el final de fichero hay dos posibilidades; si no hay fichero de salida, simplemente queremos regresar al menú; pero si sí lo hay, tenemos primeramente que cerrarlo. La alternativa habría sido bien saltar fuera del bucle al identificar la marca de final de fichero, y luego comprobar si era la opción 3; o bien, citar a otra subrutina que manejara las dos condiciones. No me gusta saltar a menos que esté absolutamente forzado a hacerlo (puede fácilmente producir código de programa que muy amablemente se describe como barroco), y usar una subrutina me parecería aquí, matar pulgas a cañonazos. (El problema surge porque el BASIC del Spectrum no tiene la cláusula ELSE en sus instrucciones IF. Algunos de los dialectos BASIC te permiten decir: IF esto THEN eso ELSE aquéllo. Lo que es algunas veces bastante útil, y significa que en el Spectrum usemos como equivalente dos instrucciones IF. Ya nos ha ocurrido varias veces, aunque este es el ejemplo más chapucero hasta ahora).

COMPARACIONES

Ya basta de preocuparse por nimiedades (el BASIC es maravilloso realmente -honestamente, ¡gracias tio Clive!)



```
870 IF ltipo THEN  GO SUB miraclavel
880 IF NOT ltipo THEN  GO SUB miraclaven
```

Ahora, realmente **sí** necesitamos un par de subrutinas más, porque el método de manejar las comparaciones numéricas y litéricas, va a ser significativamente diferente. **miraclavel** hará la comparación litérica, y **miraclaven** hará la numérica. Todo lo que necesitan entregar es un único valor "concuerda" que es 1 si se ha encontrado que son iguales, y cero en caso contrario. Por tanto, tenemos:

```
                                              coge otro registro
890 IF NOT concuerda THEN  GO TO 840
895 LET bs=comienzo: LET es=termino
900 IF opt=1 THEN  GO SUB rexreg           ] saca registro al
910 IF opt=2 THEN  GO SUB prexreg          ] aparato apropiado
920 IF opt=3 THEN  GO SUB punza            ]
925 LET comienzo=bs: LET termina=es
930 GO TO 840
                                              coge otro registro
```

Observa cómo hemos tenido que conservar "comienzo" y "término", porque se ven alterados por **exreg.**

Ahora tenemos dos pequeñas rutinas que escribir:

miraclavel	68ØØ
miraclaven	66ØØ

la **miraclavel** es más fácil, dado que no hay gamas de las que preocuparse, sólo una única clave:

```
6800 LET concuerda=0
6810 IF k$=r$(comienzo TO termino) THEN  LET concuerda=1
6820 RETURN
```

Ahora vamos con **miraclaven:**

```
6600 LET concuerda=1
6610 IF VAL r$(comienzo TO termino)<cotainf OR
        VAL r$(comienzo TO termino)>cotasup THEN  LET concuerda=0
6620 RETURN
```

Así que en **miraclavel** he supuesto que no hay concordancia, y luego ponemos concuerda a 1 si la hay; mientras que en **miraclaven** he supuesto que **sí hay** concordancia y ponemos concuerda a cero, sólo si la clave del registro de prueba es menor que la clave cota inferior de la gama o mayor que la clave superior de la gama.

Y ahora tengo que hacer una confesión. La rutina de supresión sólo funciona realmente con claves de índole litérica, tipo l; porque la línea 69Ø hace una comparación directa de literales. Probablemente pensaste sobre ello en aquel momento.

La razón de elegir hacerlo así es que **escruta** constituye una manera más potente de efectuar supresiones cuando las claves son numéricas. Después de todo, escrutar un fichero para ver una gama de claves, es exactamente lo opuesto a suprimir esas claves (usando la opción "punza" dentro de la **escruta**); así que, por ejemplo, para suprimir todos los registros con claves inferiores a 5Ø, simplemente segregamos los registros con claves mayores o igualesa 5Ø. Puedes suministrar comprobaciones y gamas más complejas de forma muy simple, ampliando **miraclaven** para incluir una segunda gama, inferior2 a superior2, digamos. De esa manera, puedes suprimir todos los registros con claves incluidas entre dos valores. (Eso también puede hacerse con la implementación corriente, pero significa generar dos subficheros).



MAS FUNCIONES

¿Qué otras opciones pudieran necesitarse, o como mínimo serían deseables? ¿Qué pasa con **listar**, que copiaría el fichero de entrada entero? Esa sería útil, pero habitualmente podemos prescindir de ella y usar la **escruta,** eligiendo una clave numérica y dándole unas cotas que sabemos sobrepasan la gama real de valores dentro del campo clave. Dado que sabemos la anchura del campo, podemos garantizar que somos capaces de hacer esto. Ahora bien, puede que quieras argüir que no es razonable esperar del usuario el uso de esa clase de artimaña, y yo estoy inclinado a creerlo así; pero esta colección de rutinas está comenzando a ser bastante grande, y si sólo tienes una memoria de 16K, el tamaño de los buzones reservados al fichero, comienza a verse adelgazado. Así que, tendríamos una buena razón para no querer ninguna nueva opción. Desde luego, si lo que tienes es una máquina de 48K, puedes continuar inventando rutinas nuevas e incluso más esotéricas, en cuanto el capricho se apodere de tí.

Hay, sin embargo, una rutina más que sin ninguna duda, debiera estar presente. Debieramos estar capacitados para sumar el contenido de un campo numérico dado dentro de cada registro. Recuerda que cuando estábamos comentando el ejemplo de la cuenta bancaria, yo sugerí que los cargos debieran meterse como negativos y los abonos como positivos! Si simplemente pudieramos sumar todos esos campos, nos daría el saldo actual. Lo que es claramente, una posibilidad provechosa.

Haremos esta opción, la 5, así que necesitamos:

```
44 PRINT AT 7,11;"5/ acumular"
```

y por tanto, la rutina comenzará a partir de la línea 1000, y empezaremos dando el valor inicial a suma:

```
1000 LET sum=0
```

Ahora determinemos cuál es la clave con la que vamos a trabajar:

```
1010 GO SUB sacaclave
```

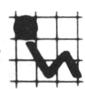
y comprobar que la clave es numérica. En caso contrario, no podríamos hacer aritmética con ella:

```
1020 IF ltipo THEN  PRINT "Clave no numerica": PAUSE 120: RETURN
```

Ahora todo lo que tenemos que hacer es coger cada registro del fichero, sumando el contenido del campo clave a cada ronda, dentro de la variable suma:

```
1030 GO SUB pinza
1040 IF r$( TO 2)="}}" THEN  PRINT "Suma de ";k$;"= ";sum:
     INPUT "Pulsa'c'para continuar";k$: RETURN
1050 LET sum=sum+VAL r$(comienzo TO termino)
1060 GO TO 1030
```

Sólo necesitamos un comentario: si se ejecutara inmediatamente un RETURN después del PRINT de la línea 1040, tendrías que ser bastante rápido para ver algo a causa del CLS que está al acecho al comienzo de la presentación del menú. Así que la instrucción INPUT, meramente provee la manera de retener el sistema hasta que el usuario esté dispuesto a continuar. La pausa de la línea 1020 es por la misma causa. De hecho, podría usarse una pausa en ambos casos, debido a que PAUSE 65535 retendría la máquina durante 21 minutos aproximadamente, o hasta que se pulsara una tecla. Así que para propósitos prácticos, el efecto es el mismo.



ADEREZANDO

Y hasta aquí, es donde voy a llevarte en esta particular excursión en el manejo de ficheros. Como ya he resaltado, no va a ser difícil añadir nuevas rutinas al sistema de gestión de datos, ni tampoco la revisión de las existentes para tener en cuenta modificaciones menores en los requisitos. Por ejemplo, cuando consigas tus microductoras de disco, será muy fácil revisar el sistema adecuadamente y el programa resultante será, desde luego, mucho más fácil de usar, dado que el Spectrum manejará y controlará el disco por sí mismo.

No hay ninguna duda que ningún trozo de programa es siempre perfecto, incluso dentro de los términos limitados que mencionamos, y ciertamente yo no reclamo que éste lo sea. Así que, haré simplemente unas pocas sugerencias para aquellas revisiones que te gustaría hacer de forma que las cosas quedaran aderezadas y maquilladas un poquito.

Primeramente, una observación sobre el propio sistema de ficheros en cassette. El sistema exige al usuario saber algo sobre el delimitador "}}". Estaremos para siempre, escribiendo:

```
IF r$ (TO 2) = "}}" THEN...
```

Podríamos incorporar esta comprobación dentro de **pinza**, eligiendo una variable llamada "fifi" (por fín de fichero, o "eof" por end of file, si eres inglés) que se pone a cero al **iniciar** y luego a 1 en **pinza** si los dos primeros octetos de r$ son "}}". Después, cualquier programa de usuario podría decir:

```
IF fifi THEN...
```

que resulta mucho más pulcro.

En segundo lugar, los mensajes presentados para que el usuario controle el cassette, están entremezclados. Sería mejor escribir separadamente cuatro subrutinas **pongra, quigra, pontoc, quitoc,** de manera que siempre se generaran los mismos mensajes en circunstancias similares. También son fáciles de revisar los mensajes, si los necesitas, o incluso sustituirlos por señales enviadas directamente a un "portal" para controlar directamente los motores del cassette, como ya sugerí anteriormente (véase Apéndice B para más detalles). Una simple revisión de los mensajes sería hacerles que parpadearan de forma que fueran más obvios; añadirles un pitido y una pausa por la misma razón (de manera que una vuelta al menú no limpiara el mensaje con excesiva rapidez).

En tercer lugar, no hay comprobación al **preparar** para asegurarse que los nombres de los campos comienzan bien con una "l" o una "n". Y obviamente eso es vital, porque el resto del sistema supone que siempre es así. Hay otras situaciones de naturaleza similar en que también debieran incluirse comprobaciones. (Por ejemplo, ¿qué sucede si das a **sacaclave** un nombre de campo que no existe?).

Finalmente, por el momento, no puedes zafarte realmente del menú. Obviamente, se hace fácilmente incluyendo una opción 6 (escape) y hacer la línea 12ØØ que sea STOP. No lo he hecho anteriormente, porque desde luego, cada vez que añades una nueva opción, la opción de salida siempre se incrementa, y la instrucción correspondiente STOP avanzaría de 2ØØ en 2ØØ.

ORDENANDO UN FICHERO

Para redondear esta sección y terminar, **te** voy a dejar con un problema.

Hay una omisión llamativa en nuestro gestor de datos. Cualquier sistema que se autorespete, debiera permitir que se pudiera ordenar el fichero según una nueva clave, de forma que por ejemplo, se tomara un fichero ordenado alfabéticamente según un campo "nombre" y se le reordenara numéricamente por, digamos, un campo "número de teléfono".



He evitado el problema, porque no es fácilmente resoluble usando una sola cinta de entrada. En los días en que las grandes instalaciones de ordenadores usaban habitualmente cinta magnética en lugar de disco, se popularizó una técnica que fue pomposamente llamada "ordenación de congregamiento polifásico".

Lo que sucedía es que tenías dos cintas de entrada y dos de salida. Leías un registro de una cinta y lo comparabas con los registros de la otra, escribiendolos en una de las cintas de salida, hasta que aparecía otro registro con una clave mayor que la del registro de referencia. En ese momento, conmutabas cintas de entrada y salida y repetías el proceso. Cuando ambos ficheros de entrada habían sido leídos completamente, usabas los ficheros de salida como nuevo grupo de ficheros de entrada y se volvía a repetir el proceso enteramente. Finalmente, después de unas tropecientas repeticiones, sólo se escribía en uno de los ficheros de salida, y la conmutación nunca tenía lugar. En ese momento, sabías que habías conseguido un fichero ordenado.

¿Suena complicado? Eso es lo que yo pienso; y ahora ya sabes por qué no he encarado ese problema hasta ahora. (Aunque en un cierto sentido, la rutina de agregamiento de registros usa una cierta clase de ordenamiento-congregamiento, usando el fichero de entrada y el buzón con los registros a agregar como equivalente de las dos cintas de entrada mencionadas; la diferencia es que esas dos entradas están separadamente ordenadas antes de que comencemos el proceso).

Pues bien, ¿es **posible** hacer el ordenamiento de un fichero en cinta con sólo una cinta de entrada? Bien, sí lo es, y hay varios algoritmos bien conocidos. Pero hay uno que descubrí recientemente que se adecúa a nuestros propósitos muy bien, porque nos permite usar la estructura de bloques que hemos incorporado en el sistema de ficheros en cassettes. (Y dije "descubierto" en lugar de "inventado" porque aunque nunca lo he visto descrito en libros, es una técnica tan simple, que estoy seguro que debe haber sido usada anteriormente).

ORDENAMIENTO POR CARRACA

Funciona como una carraca:

Tiramos del fichero de entrada, bloque a bloque, ordenando por el método de burbujas cada uno de los bloques, antes de escribirlo en el fichero de salida. Ahora bien, si eso fuera **todo** lo que hicieramos, nada sucedería que fuera muy excitante, porque aunque el fichero estaría ordenado localmente, pudiera fácilmente suceder que una clave baja al final del fichero, no se desplazaría hacia el comienzo del último bloque por muchas veces que el proceso se repitiera. De hecho, **nada** nuevo sucederá si usamos esta técnica para volver a ordenar el fichero de salida. La razón es que, dado que las particiones entre bloques permanecen en el mismo lugar, los registros sólamente pueden moverse dentro de los bloques, no entre ellos. Si pudieramos solamente cambiar las posiciones de las separaciones de los bloques...

Realmente es fácil. Antes de transferir el primer bloque al fichero de salida, escribiríamos en él, un cierto número de registros ficticios. Para lograr los mejores resultados, ese número debiera ser la mitad del número de registros de un bloque. Al leer el fichero que está situado de esta nueva forma, las fronteras entre los bloques están ahora a medio camino de sus posiciones anteriores, y este "solape" permite que tengan lugar ordenamientos posteriores. Debemos asegurarnos que los registros ficticios son ignorados por el algoritmo de ordenamiento y que se suprimen de la salida de forma que las posiciones de las fronteras de bloques, son de nuevo cambiadas para permitir otra ronda de ordenamiento. En la siguiente fase del ordenamiento se vuelven a insertar los registros ficticios, y así sucesivamente. Sabemos que hemos terminado cuando no hay posteriores cambios de posición.



Aquí hay un ejemplo que muestra el principio. Simplemente he escrito las claves para mayor simplicidad, y son los números del 1 al 12 en orden inverso. Supondremos un tamaño de bloque de 4:

12	11	10	9	8	7	6	5	4	3	2	1

En la primera pasada, ordenamos los bloques internamente e insertamos 2 registros postizos:

D	D	9	10	11	12	5	6	7	8	1	2	3	4

Segundo paso: Ordenamiento de bloques y supresión de postizos.

9	10	5	6	11	12	1	2	7	8	3	4

Tercer paso: Ordenamiento de bloques e inserción de postizos:

D	D	5	6	9	10	1	2	11	12	3	4	7	8

Cuarto paso: Ordenamiento de bloques y supresión de postizos:

5	6	1	2	9	10	3	4	11	12	7	8

Quinto paso: Ordenamiento de bloques e inserción de postizos:

D	D	1	2	5	6	3	4	9	10	7	8	11	12

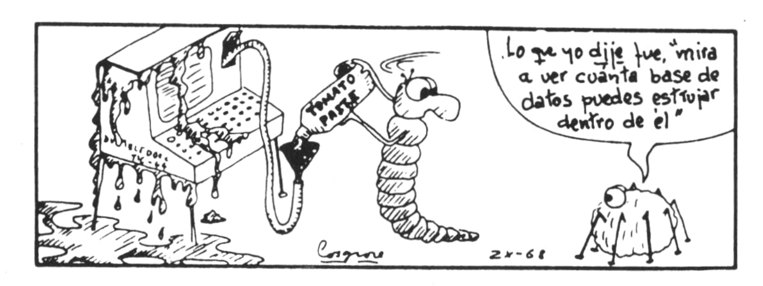
Sexto paso: Ordenamiento de bloques y supresión de postizos:

1	2	3	4	5	6	7	8	9	10	11	12

y ya lo hemos logrado!

Siempre que haya suficiente memoria libre, no hay razón para que el bloque usado para ordenamiento sea también el buzón de entrada, en cuyo caso no hay razón por la que debieramos restringirnos al tamaño del bloque para el ordenamiento del fichero. Obviamente, el número de pasos necesarios para el ordenamiento total, decrece a medida que aumenta el tamaño del bloque.

Y te dejo la codificación real del programa para que tú trabajes.





*Si piensas que CANOPO es una lata
de carne para perros, continúa leyendo...*

18 Las Cartas Astrales

Esta sección requiere bastante tiempo para imponer los datos -especialmente si decides ampliarla. Así que no la comiences a no ser que tengas un par de horas para emplear, porque es una tontería pararla en la mitad. La idea es escribir un programa que saque los mapas de las diferentes constelaciones -Orión, Cisne, Géminis, y demás. Incluye como opciones:

1. Un repaso automático de los mapas, nombrando las constelaciones.
2. Un escrutinio por nombre de una constelación dada, dibujando su forma.
3. Una prueba: la computadora dibuja una constelación y el usuario tiene que saber el nombre.

Para esta ilustración sólo pondré seis constelaciones; pero el programa estará preparado para permitir hasta 2Ø. Si quieres más de esas, guarda los datos en la cinta, vuelve a dimensionar las tablas citadas para contener los datos, y cárgalos de nuevo. (Necesitarás pensarlo todo en más detalle, pero esta es la idea general).

ESTRUCTURA DE LOS DATOS

Apaga el ordenador por un momento, porque primeramente vamos a hacer un poquito de trabajo cerebral. "Programar primero, y preguntar después" es la receta para el desastre.

Necesitamos datos para:

1. Nombre latino de la constelación (Ursa Major, etc.).
2. Nombre español de la constelación (Osa Mayor).
3. Posiciones de las estrellas (montones de coordenadas para PINTAR).
4. Magnitud (grado del brillo) de cada estrella.

Podríamos proseguir (e.g. color de la estrella, adecuadamente exagerado para una imagen bonita), pero con esos ya podemos comenzar.

Dejando a un lado por el momento, el problema de cómo conseguir la información que se precisa, (hay enciclopedias) y de cómo meterla en la máquina, debemos primeramente decidir cuál es el formato de dicha información. Obviamente, vamos a conservar los nombres en tablas; así que para 2Ø constelaciones diferentes, necesitamos preparar dos tablas litéricas de tamaño 2Ø, digamos n$ para el nombre latino y e$ para el nombre español.

También queremos escribir en pantalla cosas como:

Cygnus (el Cisne)

Ahora el problema con las tablas litéricas es que todos los literales que la forman, han de tener una longitud prefijada. Supongamos que estipulamos que sea 12 esa longitud, para permitir nombres como "Andrómeda"; y que Cygnus es el elemento número 1. No es muy satisfactorio usar la instrucción obvia:

```
PRINT n$ (1); " □ (el □ "; e$ + ")"
```

porque tendríamos algo como:

```
Cygnus □□□□□□ (el Cisne □□□□□□□□□ )
```

con todos los blancos incluidos.



Hay un truco aprovechable para evitar esto. A cada literal añadimos un carácter final, el décimo-tercero, cuyo **código** nos da la longitud real que queremos usar. Así que "Cygnus" iría como

C y g n u s □□□□□□ ✡

siendo la ✡ el carácter cuyo código es 6, la longitud de "Cygnus". (Ese es el carácter de **control** COMA PRINT, pero está bien en tanto en cuanto no intentemos exponerlo en pantalla). Para mostrar el nombre "Cygnus", usamos

PRINT n$ (1) (TO CODE n$ (1, 13))

que suprime los blancos que no queremos.

Desde luego, usamos el mismo truco en los nombres españoles; y tenemos que ser cuidadosos al preparar las rutinas de entrada y salida que tienen esta peculiaridad en cuenta.

La manera obvia de preparar las coordenadas para las estrellas de una constelación, y sus magnitudes, es también una tabla. Necesita una dimensión de 3 para permitir la coordenada horizontal, la coordenada vertical y la magnitud; siendo la otra dimensión de, digamos, 2Ø para permitir que haya 2Ø estrellas por constelación; y otra dimensión, también de 2Ø, que nos indique a qué constelación pertenece. Eso nos lleva al comando:

DIM p (3, 2Ø, 2Ø)

Sin embargo, eso exigiría 3 x 2Ø x 2Ø bloques de memoria, cada uno de 6 octetos de longitud (para un número con coma flotante), o sea, 12ØØ x 6 = 72ØØ octetos. Como un Spectrum de 16K tiene aproximadamente 9ØØØ octetos libres, después de preocuparse de los ficheros de imagen de atributos, y en el momento que el programa esté dentro junto con los otros datos, y dejemos a la máquina espacio para hacer sus cálculos... pues, con toda probabilidad, nos quedaremos cortos de memoria. Piénsalo de nuevo.

Es estupido realmente, conservar las coordenadas como puntos de comas flotantes: solamente podemos pintar con coordenadas 'x' e 'y', siendo 'x' un entero entre Ø y 255, e 'y' otro entero entre Ø y 175. Observa que Ø-255 es, por una extraña coincidencia, la gama de códigos del Spectrum. Así que, usando un solo carácter y tomando su código, podemos conseguir la coordenada 'x'. En forma similar, tanto la coordenada 'y' como la magnitud de la estrella.

Es ese caso, cada estrella ocupa tres posiciones. Veinte estrellas son 3 x 2Ø = 6Ø, que pueden empalmarse convenientemente para terminar siendo una cadena de caracteres. Para 2Ø constelaciones usamos entonces:

DIM p$ (2Ø, 6Ø)

que sólo ocupa 2Ø x 6Ø = 12ØØ octetos -una impresionante manera de estrujar las cosas.

Colocando todo este montón de cosas juntas, conseguimos la siguiente rutina de entrada. (El programa real vendrá posteriormente).

```
9000 REM Rutina de entrada, puede suprimirse despues de usada
9010 DIM n$(20,13): DIM p$(20,60)
9020 FOR i=1 TO 20
9030 INPUT 'Nombre latino?';a$
9040 LET n$(i)=a$: LET n$(i,13)=CHR$ LEN a$
9050 PRINT TAB 0;n$(i)( TO CODE n$(i,13));
9060 INPUT 'Nombre español?';a$
9070 LET e$(i)=a$: LET e$(i,13)=CHR$ LEN a$
9080 PRINT TAB 14;e$(i)( TO CODE e$(i,13))'
9090 LET c=1
9100 INPUT 'Horiz.,vertic.,magnitud ';x,y,m
9110 IF x=0 THEN  GO TO 9150
9120 LET p$(i)(3*c-2 TO 3*c)=CHR$ x+CHR$ y+CHR$ m
9130 LET c=c+1: IF c>20 THEN  GO TO 9150
9140 GO TO 9100
9150 NEXT i
```




LOS DATOS

El siguiente paso a dar es meter los datos dentro de la máquina. Explicaré posteriormente cómo desarrollarlo para otras constelaciones; y puede saltarse esta sección por el momento, para ver primero el resto del programa.

Rula la rutina de entrada, y teclea todo lo siguiente cuando te lo solicite. Cada línea representa la respuesta a una pregunta de la máquina, observa que necesitas pulsar ENTER después de las tres partidas que corresponden a una estrella. El formato de la pantalla no será exactamente el mismo. Teclea Ø Ø Ø para parar.

Cancer - **Cangrejo**

95	54	4
111	141	4
113	87	4
115	96	4
149	42	4
156	89	4

Cygnus - **Cisne**

64	12Ø	4
72	36	3
74	85	4
76	76	4
91	1Ø7	4
1Ø4	54	2
1Ø5	66	4
114	111	1
122	36	4
124	86	2
139	123	4
142	118	4
161	5Ø	4
168	1ØØ	3
179	146	4
188	156	4
192	26	3

Gemini - **Los gemelos**

66	113	1
68	94	3
76	1Ø6	4
78	132	1
88	111	4
97	82	3
1ØØ	54	3
1Ø5	123	4
118	76	4
124	144	3
14Ø	1ØØ	3
143	36	3
152	54	2
162	76	4
168	87	3
178	89	3
193	94	4

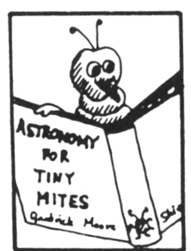


Leo - **León**

28	90	2
56	49	4
58	74	4
72	97	3
76	122	2
78	53	4
92	118	4
100	139	4
120	52	4
144	110	2
150	128	3
153	57	1
158	91	3
181	140	4
188	126	3
188	52	4
204	121	4
216	152	4

Orion - **Cazador**

78	133	4
86	121	1
104	32	2
108	72	2
110	106	4
113	136	3
115	76	2
118	51	3
120	81	2
126	117	2
128	94	4
130	71	3
134	44	4
142	50	3
144	100	4
148	43	1
154	100	3
154	60	3
171	116	4
172	124	3











Ursa Major - **La Osa Mayor**

27	131	2
56	140	2
73	134	2
94	128	3
104	116	2
106	87	4
124	21	3
126	15	4
133	70	3
136	142	2
137	118	2
173	60	3
176	66	3
176	140	4
180	156	4
183	117	4
200	112	3
211	162	3
225	103	3
226	109	3

Ahora, si todavía continúas conmigo, guarda este trozo en la cinta antes de que suceda algún percance. La forma más fácil es guardar todo, incluyendo la rutina de carga, eso tiene la ventaja de que puedes querer cargar algo más posteriormente. O puedes usar **el almacenamiento de tablas:** que se hace en tres pasadas, usando:

```
SAVE "Latin" DATA n$ ( )
SAVE "Español" DATA e$ ( )
SAVE "Estrellas" DATA p$ ( )
```

que pueden ser recuperados de la cinta usando instrucciones similares, pero cargando en lugar de guardando.

COMPROBACIONES Y CORRECCIONES

Si cometes un error mientras estás haciendo esto, no se ha perdido todo. Supón que la quinta estrella en Leo está equivocada. Puedes teclear el comando directo:

```
LET i = 4: LET c = 5: GO TO 9100
```

luego tecleas el valor correcto para x, y, m: y finalmente STOP. Para comprobar lo metido, utiliza esta rutina:

```
9500 FOR i=1 TO 20
9510 PRINT n$(i)( TO CODE n$(i,13))
9520 PRINT e$(i)( TO CODE e$(i,13))
9530 FOR t=1 TO 60 STEP 3
9540 IF p$(i,t)="0" OR p$(i,t)="□" THEN  GO TO 9570
9550 PRINT CODE p$(i,t);"□";:
     PRINT CODE p$(i,t+1);"□";:
     PRINT CODE p$(i,t+2)
9560 NEXT t
9570 NEXT i
```



Sólo úsala si estás preocupado sobre la exactitud: si has comprobado la imagen en la pantalla durante la pasada de **imputación de datos** no sería necesario.

Si tienes impresora, cambia cada PRINT a LPRINT, y tendrás una copia en papel permanente para tu referencia.

ELABORANDO LOS DATOS

Si quieres añadir otras constelaciones, debes determinar qué números vas a imputar.

La manera más simple es dibujar la constelación en un trozo de papel reticulado, a poder ser con la retícula de 256 x 176 marcada sobre él. (Realmente yo usé una retícula de 64 x 44 y multipliqué por 4). Comienza con un libro sobre astronomía: **Norton's Star Atlas,** publicado por Call e Inglis, es bueno. Copia las estrellas sobre un papel transparente, y transfiere los resultados a tu retícula; y cuidadosamente saca las coordenadas, comprobando el atlas de las estrellas para hallar la magnitud. Mira la figura 18.1 para los datos de Leo.

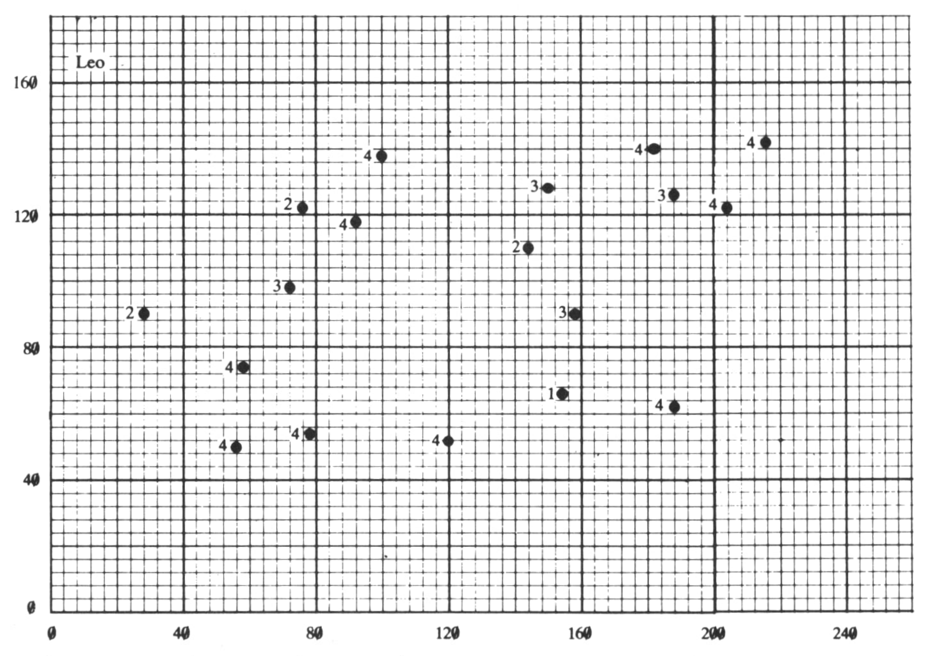


Figura 18.1 Datos de entrada para Leo.





Necesitarás escribir una pequeña rutina para transferir las coordenadas en pantalla a p$, y para asignar adecuadamente la magnitud. Es un proyecto para tí si estás dispuesto. La variable sistemal COORDS (Capítulo 6) puede demostrar su utilidad.

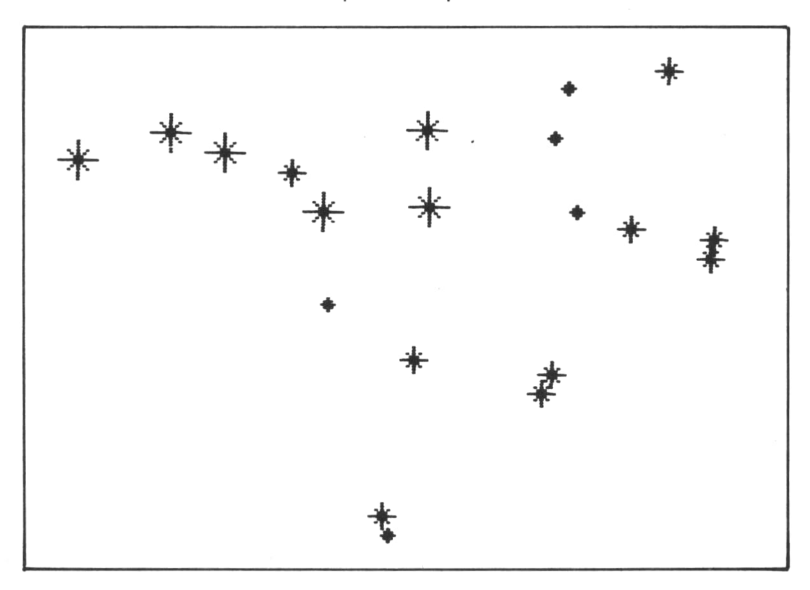


Figura 18.2 Imagen de la Osa Mayor. Véase el famoso Carro arriba a la izquierda.

EL PROGRAMA PRINCIPAL

Puedes suprimir las rutinas de carga y comprobación si lo deseas. Simplemente **no teclees** RUN, o perderás los datos y tendrás que volver a recuperarlos de la cinta. Usa en su lugar GO TO 1.

Obviamente, necesitamos una rutina que muestre las estrellas. Esta rutina pinta una estrella, cuyo tamaño está determinado por la magnitud m, y la pinta en la posición x, y. (Las estrellas de magnitud 1 son las más brillantes, luego la 2, luego la 3, etc.) La constelación es el número i:

```
8000 REM Pinta estrellas
8010 PAPER 0: INK 7: BORDER 0: CLS
8020 FOR t=1 TO 60 STEP 3
8030 LET k=CODE p$(i,t)
8040 IF k=32 OR k=48 THEN  RETURN
8050 LET x=k: LET y=CODE p$(i,t+1):
     LET m=CODE p$(i,t+2)
8060 GO SUB 8500
8070 NEXT t
8080 RETURN
8500 REM Dibuja una estrella
8510 LET s=10-2*m
8520 PLOT x,y-s: DRAW 0,2*s
8530 PLOT x-s,y: DRAW 2*s,0
8540 PLOT x-s/2,y-s/2: DRAW s,s
8550 PLOT x-s/2,y+s/2: DRAW s,-s
8560 RETURN
```



Ahora, si queremos tres opciones: lista automática, búsqueda por nombre y prueba de conocimiento. Vamos a preparar un pequeño menú:

```
100 CLS : PRINT 'Mapas estelares'
110 PRINT 'Opciones:
                    1. Lista Automatica
                    2. Busqueda por nombre
                    3. Prueba de Astronomia'
120 INPUT 'Numero de la opcion?';opt
```

Ya eres veterano en este juego, así que te dejamos la tarea a tus gustos personales.

```
130 GO SUB 1000*opt
140 PAUSE 0: CLS : GO TO 100
```

Ahora escribimos las opciones:

```
1000 FOR i=1 TO 6
1010 GO SUB 8000
1020 PRINT AT 0,0;n$(i)( TO CODE n$(i,13));
     '□(';e$(i)( TO CODE e$(i,13));')'
1030 INPUT 'Pulsa ENTER para continuar';d$
1040 NEXT i
1050 RETURN
2000 INPUT 'Nombre latino de la constelacion?';q$
2010 FOR i=1 TO 6
2020 IF n$(i)( TO CODE n$(i,13))<>q$ THEN  GO TO 2045
2030 GO SUB 8000
2040 PRINT AT 0,0;q$;'□(';e$(i)( TO CODE e$(i,13));')'
2045 NEXT i
2050 RETURN
3000 LET i=INT (1+6*RND)
3010 GO SUB 8000
3020 INPUT 'Que constelacion es esta?';q$
3030 IF q$=n$(i)( TO CODE n$(i,13))
     THEN  PRINT AT 0,0; FLASH 1;'MUY BIEN!'
3040 IF q$<>n$(i)( TO CODE n$(i,13))
     THEN  PRINT AT 0,0; FLASH 1;'Lo siento, no es correcto':
     PRINT AT 1,0; FLASH 0;n$(i)( TO CODE n$(i,13))
3050 RETURN
```

Para ensayar con esto, teclea GO TO 1 (recuerda, **no** RUN), y sigue las instrucciones de la pantalla.

Si has puesto más de 6 constelaciones, necesitas cambiar el valor 6 de las líneas 1ØØØ, 2Ø1Ø y 3ØØØ según el nuevo número.

Deposita en cinta todo esto, usando algo como:

SAVE "Mapastelar" LINE 1ØØ

y a continuación, rulará automáticamente, inmediatamente después de ser cargado, evitando el peligro de borrar todas las variables si tienes la costumbre de teclear RUN.



SOLUCIONES A LOS PROBLEMAS DE CRIPTOANALISIS (Pág. 106)

1. abcdefghijklmnopqrstuvwxyz
   zvetrsnbjpkcuxdfgayohlqimw

   Noeselhombreelquehablasinoellenguajeatravesdelhombre

2. abcdefghijklmnopqrstuvwxyz
   mlvroazkegfudwjyqbxtpsicnh

   Elhombreestahabituadoaanalizarantetodolo
   sproblemasyenunsegundoperiodoabuscarless
   olucionconelinstrumentodelrazonamientologico

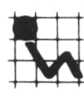
3. abcdefghijklmnopqrstuvwxyz
   qcynrzemsokawhfdxblguptvij

   Esimprescindiblequeelhombrepropongalosme
   diosparaqueenelfuturopuedavivirenambitoe
   cologicogratodondepuedadesplegartodassusposibilidades

4. abcdefghijklmnopqrstuvwxyz
   bmflrnywjgahiepxqdkozcsutv

   Puestoquecadaseresbiologicaysocialmenteu
   nicodebemosatenderledemanerapprogresivame
   ntepersonalizada

Desde luego, no serás capaz de determinar toda la tabla de código: sólo las letras que realmente **hemos usado.**



Apéndices



APENDICE A: El Sistema de Ficheros en Casette
Una descripción para referencia

Aunque en este libro, lo hemos usado únicamente dentro del Gestor Spectrum de Información, no hay desde luego, ninguna razón por la que no pudieramos incorporarlo a otros programas. Para hacerlo, simplemente tienes que meter todas las rutinas conjuntamente con sus identificadores y números de línea en la que comienzan, y depositarlo en cinta como un programa llamado "cfs". Cuando tengas en memoria el programa con el que quieres usar **cfs**, simplemente teclea:

```
MERGE cfs
```

y pon en marcha la cinta, como si estuvieras haciendo una carga normal. El efecto es **congregar** los dos programa, el que hay en memoria con el recuperado de la cinta, y conseguir de una forma fácil agregar **cfs** (o cualquier otro utensilio de programación) a cualquier programa.

Desde luego, el programa principal no debe usar ninguno de los números de línea que use el programa agregado, y debe procurarse que no haya ningún conflicto en el uso de los nombres de las variables (en caso contrario, sería fácil hacer que q valiera 1; citar a una subrutina, y encontrarte que misteriosamente q ha pasado a valer 14). Este apéndice provee la información necesaria para evitar esos conflictos.

Doy un listado completo de **cfs**, junto con las descripciones de las gestiones que cada rutina lleva a cabo, y de los nombres de las variables que usa. Observa que el listado **no** es idéntico al dado en el texto. Se han hecho las mejoras que hemos comentado en algún lado, y también hemos renumerado algo.

NUMEROS DE LINEA

cfs usa la línea 1 y todos los números de línea a partir de la 9ØØØ.

NOMBRE DE LAS VARIABLES

Las variables usadas por **cfs** pueden clasificarse en cuatro clases:

1. **Global**

Una variable global es una que se usa en varias de las rutinas **cfs**, y consecuentemente nunca debe ser redefinida en el programa usuario de las rutinas.

Por ejemplo, **iniciar** prepara una tabla llamada n$ que contiene los nombres de los campos en los ficheros. Si el usuario la redefine en alguna parte de su programa, todos los nombres de las variables se convertirán inmediatamente en ristras de blancos.

2. **Local**

Son variables que sólo se utilizan dentro de una rutina **cfs**, pero que no tienen ningún significado fuera de ella. El programa de usuario puede usar esos nombres, siempre y cuando no necesite que sus valores se mantengan antes y después de citar a una rutina **cfs** que las usa localmente. Por ejemplo, **inreg** usa p y c como contadores de bucles. Si escribimos:

```
FOR p = 1 TO 4
GO SUB inreg
NEXT p
```

ese bucle solamente efectuará una ronda si el bucle interno de **inreg** que usa la p, se ha ejecutado cuatro o más veces!.



Pero escribiendo:

```
FOR p = 1 TO 4
  cualquier otra cosa
NEXT p
GO SUB inreg
```

es perfectamente válido y funciona.

3. Parámetros entregados a las rutinas cfs
Son los nombres de las variables que sirven de argumento para la rutina **cfs.** Por ejemplo, r$ debe tener el valor preparado al citar **punzar** para que su contenido sea grabado en la cinta.

4. Parámetros devueltos por las rutinas cfs
Son las variables que la rutina **cfs** ha elaborado como resultado para ser usados subsecuentemente por el programa usuario. Por ejemplo, r$ es devuelto por **pinzar,** con los datos "lectados" del cassette.

VARIABLES GLOBALES

Nombre	Uso
lpr	Longitud del registro en octetos.
fifi	final de fichero: Ø = no final, 1 = final
f$	Nombre del fichero de entrada
g$	Nombre del fichero de salida
pin	puntero al siguiente registro en el buzón de entrada
i$ ( )	Tabla de 2D que actúa como buzón de entrada. Su tamaño se determina por w (1) y lpr
inbc	Número del bloque de entrada corriente.
n$ (11, 1Ø)	Tabla con los nombres de hasta 1Ø campos por registro. Cada nombre de campo puede tener hasta 1Ø octetos.
pex	Puntero al siguiente registro en el buzón de salida.
e$ ( )	Tabla de 2D que actúa como buzón de salida. El tamaño es igual que para i$.
exbc	Número del bloque de salida corriente.
testigo	Puesto a cero cuando no está activado el cassette; puesto a 1 si el cassette está activado.
w (11)	Tabla con las anchuras de los campos, en octetos, correspondientes a los campos con nombre en n$. w (1) contiene el número de registros por bloque.

DESCRIPCIONES DE LAS RUTINAS

cierre

Acción	Graba marcas de final de fichero en el buzón de salida.
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	Ninguna (pero nota que r$ es sobreescrita)
Variables locales	Ninguna
rutinas cfs citadas internamente	punzar



Listado

```
9740 LET r$=" }} ":
9750 GO SUB punza
9760 LET r$="ultimal"
9770 GO SUB punza
9780 RETURN
```

cogebloque

Acción	Mete un bloque del fichero de entrada
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	inbc, i$, pin
Variables locales	m$
rutinas cfs citadas internamente	avipin

Listado

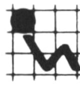
```
9800 LET m$=STR$ inbc
9810 GO SUB avipin
9820 LOAD f$+m$ DATA i$()
9825 POKE 23692,255
9830 GO SUB avipin
9840 LET pin=1
9850 LET inbc=inbc+1
9860 RETURN
```

iniciar

Acción	Inicializa el sistema y establece los nombres y descripciones de los ficheros. Los ficheros postizos se denominan "nulo"
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	n$, w, pin, pex, inbc, exbc, lpr, fifi, testigo, f$, g$, i$, e$
Variables locales	p
rutinas cfs citadas internamente	prepare, avipin, avipun, arredre

Listado

```
9500 DIM n$(11,10): DIM w(11): LET lpr=0: GO SUB arredro
9505 INPUT "Nombre fichero de entrada:        ";f$
9510 IF f$="nulo" THEN  GO SUB prepare: GO TO 9525
9514 GO SUB avipin
9515 LOAD f$+"h1" DATA n$()
9520 LOAD f$+"h2" DATA w()
9521 GO SUB avipin
9525 INPUT "Nombre fichero de salida:         ";g$
9530 FOR p=2 TO 11
9535 LET lpr=lpr+w(p)
9540 NEXT p
9545 DIM i$(w(1),lpr): DIM e$(w(1),lpr)
9547 IF g$="nulo" THEN  RETURN
9548 GO SUB avipun
```



```
9550 SAVE g$+"h1" DATA n$()
9555 SAVE g$+"h2" DATA w()
9557 GO SUB avipun
9560 RETURN
```

inreg

Acción	Avisa para que impongas por teclado un registro, campo a campo, y coloca el resultado dentro de r$
Parámetros recibidos	Ninguno
devueltos	r$
Variables globales afectadas	Ninguna
Variables locales	a$, comienzo, p, c, término
rutinas cfs citadas internamente	Ninguna

Listado

```
9000 DIM a$(lpr)
9010 CLS : LET comienzo=1
9020 FOR p=2 TO 11
9025 IF n$(p,1)="□" THEN  GO TO 9120
9030 PRINT AT p,0;n$(p,1);":";AT p,4;n$(p)(2 TO )
9040 FOR c=1 TO w(p)
9050 PRINT AT p,14+c;"_"
9060 NEXT c
9070 LET termino=comienzo+w(p)-1
9080 INPUT (n$(p)(2 TO ));a$(comienzo TO termino)
9090 PRINT AT p,15;a$(comienzo TO termino)
9100 LET comienzo=termino+1
9110 NEXT p
9120 LET r$=a$
9130 RETURN
```

avipin

Acción	Avisa que pongas o quites el cassette en el modo "pinzado de cinta"
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	testigo
Variables locales	Ninguna
rutinas cfs citadas internamente	Ninguna

Listado

```
9140 PRINT  INVERSE 1;"Marche" AND NOT testigo;
          "Quiete" AND testigo;" Rascando"
9150 BEEP .2,15: PAUSE 6: BEEP .3,20: PAUSE 80
9160 LET testigo=NOT testigo
9170 RETURN
```




avipun

Acción	Avisa que pongas o quites el cassette en el modo "punzado de cinta"
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	testigo
Variables locales	Ninguna
rutinas cfs citadas internamente	Ninguna

Listado

```
9300 PRINT  INVERSE 1;"Marche" AND NOT testigo;
        "Quiete" AND testigo;" Grabando"
9310 BEEP .2,20: PAUSE 6: BEEP .3,5: PAUSE 80
9320 LET testigo=NOT testigo
9330 RETURN
```

exreg

Acción	Expone en pantalla los campos del registro r$
Parámetros recibidos	r$
devueltos	Ninguno
Variables globales afectadas	Ninguna
Variables locales	p, comienzo, término
rutinas cfs citadas internamente	Ninguna

Listado

```
9200 LET comienzo=1
9210 FOR p=2 TO 11
9215 IF n$(p,1)="□" THEN  PRINT : RETURN
9220 PRINT n$(p,1);":";TAB 4;n$(p)(2 TO );
9240 LET termino=comienzo+w(p)-1
9250 PRINT TAB 15;r$(comienzo TO termino)
9260 LET comienzo=termino+1
9270 NEXT p
9280 PRINT : RETURN
```

dejabloque

Acción	Deposita en la cinta un bloque de registros
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	exbc, pex
Variables locales	m$
rutinas cfs citadas internamente	Ninguna

Listado

```
9900 LET m$=STR$ exbc
9910 GO SUB avipun
9920 SAVE g$+m$ DATA e$()
9930 GO SUB avipun
9940 LET pex=0
9950 LET exbc=exbc+1
9960 RETURN
```



pinza

Acción	Capta el siguiente registro del buzón de entrada y lo coloca en r$
Parámetros recibidos	Ninguno
devueltos	r$
Variables globales afectadas	pin, fifi
Variables locales	Ninguna
rutinas cfs citadas internamente	cogebloque

Listado

```
9600 IF pin=0 OR pin>w(1) THEN  GO SUB cogebloque
9610 IF i$(pin)( TO 6)="ultimal" THEN
     PRINT "Intento de sobrepasar fifi": STOP
9620 LET r$=i$(pin)
9625 IF r$( TO 2)=" }} " THEN  LET fifi=1
9630 LET pin=pin+1
9640 RETURN
```

Comentario: Observa que **fifi** vuelve a ponerse a cero en **arredre.** Así que si va a volverse a tratar un fichero de entrada sin hacer una segunda cita a **iniciar**, y se comprueba el fin de fichero mediante la forma IF fifi THEN... es necesario meter GO SUB **arredre** al comienzo del segundo tratamiento del fichero. Así se ponen a cero también los punteros y contadores.

arredre

Acción	Vuelve hacia atrás los punteros de buzones, los testigos, etc.
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	pin, pex, inbc, exbc, fifi, testigo
Variables locales	Ninguna
rutinas cfs citadas internamente	Ninguna

Listado

```
9970 LET pin=0: LET pex=0: LET inbc=0: LET exbc=0:
     LET fifi=0: LET testigo=0
9980 RETURN
```

prepare

Acción	Permite al usuario definir la forma de los registros y de los bloques
Parámetros recibidos	Ninguno
devueltos	Ninguno
Variables globales afectadas	n$, w
Variables locales	nf, p
rutinas cfs citadas internamente	Ninguna



Listado

```
9400 INPUT "Numero de campos: ";nf
9405 CLS
9410 FOR p=2 TO nf+1
9415 PRINT AT 10,2;"Campo□";p-1
9420 INPUT "Nombre del campo  (precedido de 1/n):";n$(p)
9422 IF n$(p,1)<>"1" AND n$(p,1)<>"n" THEN  GO TO 9420
9425 INPUT "Numero de octetos: ";w(p)
9430 NEXT p
9435 CLS
9440 INPUT "No. de registros por bloque: ";w(1)
9445 RETURN
```

punza

Acción	Larga el registro r$ al buzón de salida
Parámetros recibidos	r$
devueltos	Ninguno
Variables globales afectadas	pex, e$
Variables locales	Ninguna
rutinas cfs citadas internamente	dejabloque

Listado

```
9700 LET pex=pex+1
9710 LET e$(pex)=r$
9720 IF pex=w(1) OR r$="ultimal" THEN  GO SUB dejabloque
9730 RETURN
```

INICIALIZACIONES EN LA LINEA 1

```
1 LET cierre=9740:      LET cogebloque=9800: LET iniciar=9500:
  LET inreg=9000:       LET avipin=9140:     LET avipun=9300:
  LET rexreg=9200:      LET dejabloque=9900: LET pinza=9600:
  LET arredro=9970:     LET prepare=9400:    LET punza=9700
```



APENDICE B: Control Automático del Cassette

Ya se ha mencionado que, en principio, no hay dificultad en controlar automáticamente los motores del cassette. Esta es una técnica específica para hacerlo.

Un **portal** de entrada/salida aconsejable, es el circuito integrado ZX Spectrum, comercializado por Kempston (Micro) Electronics. Tiene tres portales de 8 bits dentro de él que pueden ser programados para actuar como portales de entrada o de salida,o combinaciones de ambos. En lo que a nosostros nos concierne, sólo se necesitan 2 bits de los 24 disponibles. Así que elegirémos los bits bajos (bit Ø) de los portales B y C para controlar respectivamente los mandos del cassette PLAY y RECORD. (Lo hacemos porque las conexiones a los enchufes para esos portales son los mismos, mientras que para el portal A son ligeramente diferentes).

Desde el portal, no se puede excitar directamente un relé, dado que sólo se pueden emplear pequeñas corrientes (niveles TTL). Así que la salida del circuito se emplea en conmutar un transistor, tal y como se muestra en la figura B1.

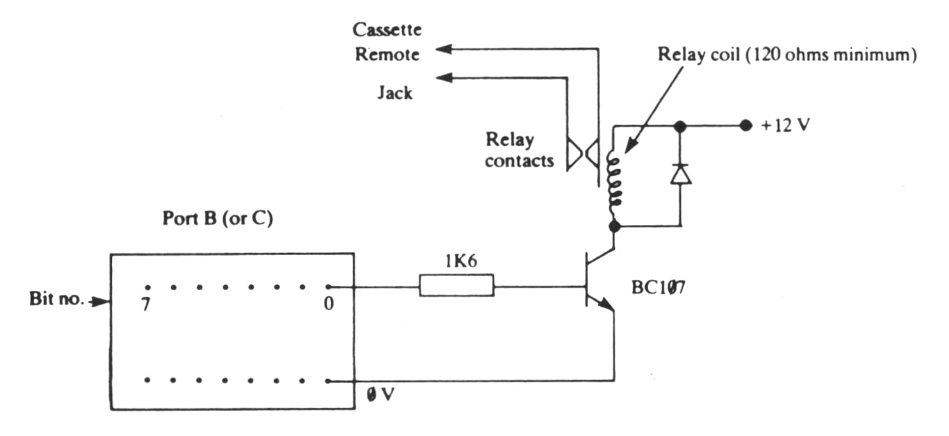


Figura B1

Los portales se preparan en el modo de salida mediante la instrucción:

```
OUT 127, 128
```

Que puede insertarse al comienzo de la rutina **iniciar.**

Luego es necesario poner a 1 el bit Ø del PORTAL B para activar el cassette en el modo de reproducción, y poner a 1 el bit Ø del PORTAL C para activarlo en el modo de grabación. Para hacerlo, simplemente sustituye las subrutinas **avipin** y **avipun** tal y como sigue:

```
avipin: 914Ø   LET testigo = NOT testigo
        915Ø   OUT 63, testigo
        916Ø   RETURN

avipun: 93ØØ   LET testigo = NOT testigo
        931Ø   OUT 95, testigo
        932Ø   RETURN
```

siendo 63 y 95 las direcciones asignadas a los portales B y C, respectivamente.

El circuito de la figura B1 está basado en uno diseñado por Kempston (Micro) Electronics cuya dirección es 60, Adamson Court, Hillgrounds Road, Kempston, Bedford MK42 8QZ.



APENDICE C: Una Guía para el Usuario de SDM
El Gestor Spectrum de Información

SDM es un sistema simple de gestión de ficheros, que permite el uso de un fichero de entrada y un fichero de salida. Ambos ficheros pueden especificarse como no existentes si se les da el nombre reservado "nulo". Los registros son de longitud prefijada y constante, y pueden constar de 1Ø campos, teniendo cada uno un nombre y una longitud definida por el usuario. El nombre de un campo consta de 1Ø caracteres como máximo, el primero de los cuales debe ser "l" o "n" (sólo minúsculas) para indicar si el campo es de índole litérica o numérica. Esta distinción es importante, porque determina cómo se van a efectuar las adiciones de registros al fichero y la escruta del fichero en busca de un registro. Por ejemplo, un intento de segregar aquellos registros cuyo tercer campo contiene el valor 357, no encontrará un registro cuyo tercer campo 357.Ø o +357, si ese campo ha sido declarado de índole "l". Además, es ilegal intentar sumar campos cuya índole es "l". Por eso los llamamos litéricos.

Los nombres del fichero están definidos por el usuario. Deben seguir las reglas normales de los nombres de ficheros en BASIC, excepto que su longitud no debe sobrepasar los 8 caracteres (2 menos que la restricción en BASIC). Si el fichero va a ser muy largo, puede ser más seguro imponer un límite de, digamos, 6 caracteres por nombre. La razón es que el nombre de fichero que finalmente se pasa a BASIC, está formado a partir del nombre del fichero dado por el usuario más un número de bloque correspondiente a esa sección. De hecho, los bloques de un fichero que se llame "nomefich" son:

nomefichh1
nomefichh2
nomefichØ
nomefich1
nomefich2
etc.

Los primeros dos bloques, forman la cabecera del fichero y contienen la descripción para el sistema, de los registros y la estructura de los campos de ese fichero. A continuación vienen los bloques de datos, que son numerados secuencialmente a partir de cero. Consecuentemente, un fichero que tenga más de 1ØØ bloques y un nombre de 8 posiciones, vulnerará las reglas del BASIC, y el programa se "rumpirá" con error F. En la práctica, sin embargo, no es probable un fichero de ese tamaño.

Las operaciones permitidas por SDM son:

1. La creación de ficheros.
2. La modificación de los ficheros existentes agregando o suprimiendo registros, o mediante la creación de subficheros, segregados del fichero principal.
3. La escruta de ficheros, en busca de registros que tengan determinados atributos.
4. La totalización del valor de un determinado campo de todos los registros del fichero.

Estas operaciones son accesibles mediante un menú que se muestra después de haber concluido una de las operaciones. Observa, sin embargo, que la repetición del menú es para facilitar diversas operaciones sobre el **mismo** fichero. Si las operaciones a realizar son sobre ficheros diferentes, el programa debe ser "rulado" de nuevo.

ACCION DE SDM

Después de haber cargado SDM y haber tecleado RUN, aparece en pantalla el mensaje "Gestor de Datos Spectrum", seguido de un aviso solicitando el nombre del fichero de entrada. Si tuvieras tal fichero, depositado en cassette, deberías introducir la cinta en el cassette y teclear el nombre correspondiente. El sistema te solicitaría a continuación, que activaras el cassette para poder recuperar los dos bloques de cabecera. Luego te pediría que apagaras el cassette.



Si no hay fichero de entrada, debieras teclear la palabra "nulo". El sistema te requeriría después la definición del fichero mediante los siguientes avisos.

	Aviso	Significado
	1. Nº de campos	Número de campos de un registro.
Estos pasos se repiten para cada campo. El número asignado al campo dentro del registro se muestra en un aviso adicional.	2. Nombre del campo (primer caracter: l/n)	Teclea el nombre del campo precedido por l o por n, y que conste de 9 caracteres como máximo. e.g. un campo que contenga un nombre puede ser "lnome", uno que sea una cuenta bancaria puede ser "nbal".
	3. Nº de octetos	Es el número máximo de octetos que va a ocupar cualquier valor de los asignados a ese campo cuyo nombre acabas de imponer.
	4. Nº de registros por bloque	Para simplicidad de la operación, debiera elegirse suficientemente grande, típicamente 2Ø o más. Sin embargo, las restricciones de memoria y del tamaño de cada registro, crean una barrera práctica para tamaños grandes de bloque, en especial en una máquina de 16K. Los tamaños máximos para el bloque pueden fácilmente calcularse en un caso dado, pero habitualmente es más fácil elegir el valor por ensayo y error, asegurándose siempre que no aparece el código 4 "Out of memory".

El sistema requiere luego el nombre del fichero de salida. Si no va a generarse ningún nuevo fichero, debiera teclearse "nulo". En caso contrario, debiera teclearse un nombre según las mismas reglas que antes. El sistema te pedirá que actives el cassette en el modo de grabación, y depositará en la cinta los bloques de cabecera, después de lo cual, te avisará para que apagues el cassette.

El menú que ahora aparece, se muestra debajo:

Las opciones son:

1. crear
2. agregar
3. suprimir
4 escrutar
5. acumular
6. escapar

y se espera que el usuario elija una de esas opciones mediante el número asociado.

Cada opción se describe ahora por separado.

1. crear

Se avisa la usuario que teclee cada uno de los campos de un registro sucesivamente. El tamaño permitido para el campo se indica por el número de posiciones subrayadas que aparecen frente al nombre del campo.



(El tipo de campo se separa del resto del nombre por dos puntos, como en n:bal). A medida que se impone cada campo, el registro se construye en la pantalla, de manera que el usuario pueda comprobar lo impuesto. Cuando está completo un registro, aparece el aviso "Alguno más (s/n)". Si hay más registros a imponer, el usuario teclea "s"; si no "n". En el primero de los casos, se repetirá el proceso; en el segundo caso se avisará al usuario que ponga el cassette en el modo de grabación y se cerrará el fichero. (Observa que la creación del fichero estará ayudada de los avisos oportunos para controlar el cassette a medida que se llenan bloques).

2. agregar

Se pregunta al usuario el número de registros que desea agregar al fichero; a continuación se imponen por teclado los valores en la misma manera que para la rutina **crear** (exceptuando que el proceso se repite el número de veces que se ha designado). Se pueden imponer los registros a agregar en cualquier orden.

Al usuario se le pregunta a continuación, el nombre del campo clave. Los valores de ese campo son los que definen el orden en que deberán colocarse los registros dentro del fichero. Ese nombre **no** debe incluir la índole del campo (e.g. teclea "bal" y **no** "nbal"). Los nuevos registros se insertan automáticamente después y en su sitio, con el sistema avisando al usuario de vez en cuando para que controle el cassette.

3. suprimir

Se pregunta primeramente al usuario, el número de registros de cada tipo que quiere suprimir, y luego el nombre del campo clave (excluyendo el código correspondiente a la índole del campo).

Finalmente, se le pide que teclee cada una de las claves asociada a cada uno de los registros a suprimir.

Por ejemplo, supongamos que deseamos suprimir los registros de Pepe Pérez, Paco López y Luis García. Teclearemos:

```
¿Cuántos registros? 3
Teclea nombre del campo clave: nome

Teclee supresión (sólo clave): Pepe Pérez
Teclee supresión (sólo clave): Paco López
Teclee supresión (sólo clave): Luis García
```

El sistema responderá luego de manera similar al de la rutina **agregar.**

4. escrutar

Se pregunta primero al usuario el nombre del campo clave (cuyos valores van a ser examinados). Si es una clave de índole l, la pregunta siguiente es sobre la clave "objetivo". Por ejemplo, si existe un campo llamado **deducible** cuyo contenido será "s" si el importe que haya en otro campo del registro es deducible de impuestos, y quieres listar todos los registros con esa particularidad, las respuestas a las preguntas serán:

```
Teclea nombre campo clave:  deducible
Clave objetivo: s
```

Si el campo clave es de índole numérica, la segunda pregunta solicita una **gama,** mediante la cota superior y la cota inferior del valor del campo. Si sólo se quiere un valor, basta contestar con ese valor para ambas cotas. Por ejemplo, si hay un campo llamado **nimporte,** y deseamos examinar todos los registros en que ese importe valga 1ØØ, las preguntas aparecerán como:

```
Teclee nombre campo clave: importe
Gama de clave -inferior: 1ØØ
              -superior: 1ØØ
```



Si, sin embargo, deseamos buscar todos los valores mayores o iguales a 100, podríamos teclear:

Gama de clave -inferior: 100
superior: 1000

siempre y cuando supiéramos que no pueden existir valores mayores de 999. Siempre podremos elegir un valor así porque sabemos la longitud del campo, de manera que la especificación anterior está garantizado que es correcta si **nimporte** tiene una longitud de 3 octetos. Igualmente, si quisiéramos todos los registros con valor inferior a uno dado, debiéramos fijar la cota inferior al valor mínimo posible (e.g. -100 para un campo de 3 octetos).

Se pregunta finalmente al usuario a qué aparato quiere enviar la salida; puede ser la pantalla, la impresora o un fichero de salida en cassette.

La última opción permite segregar subficheros del fichero principal (e.g. se puede crear así un nuevo fichero con aquellos registros que son deducibles de impuestos).

Observa que aunque no hay una opción para listar todo el fichero por impresora, puedes usar **escrutar** para remedar esa función usando cualquier clave numérica, cuyas cotas inferior y superior estén fuera de la gama permitida.

5. acumular

Se pregunta al usuario el nombre del campo cuyo contenido va a ser sumado para todos los registros del fichero. Obviamente el campo debe ser numérico, por lo que en caso contrario te muestra un mensaje de error y el sistema regresa al menú principal. La cantidad total final, se expone en el monitor, y el sistema espera ahí hasta que pulses una tecla, antes de regresar al menú.

COMENTARIOS GENERALES

Todas las operaciones standard, se efectúan sobre el fichero completo. Por lo tanto, una pregunta tal como "¿Cuál es el total de todos los registros deducibles de impuestos?" no se puede contestar directamente. Sin embargo, se puede solventar el problema creando un fichero con los registros que tienen la particularidad de ser deducibles usando escruta de ficheros, y luego acumulando los importes de los campos de ese nuevo fichero.

Igualmente, escrutinios más complejos (ejemplo: ¿cuántos registros deducibles de impuestos tienen sus importes comprendidos entre 2000 y 20000 ptas.?) puede efectuarse haciendo sucesivas escrutas y generando nuevos subficheros en cada ronda.



APENDICE D: Gestor de Datos Spectrum - Listado del Programa

El listado final revisado se muestra a continuación. Por simplicidad, incluye las rutinas **cfs**; que supongo ya has guardado previamente, por lo que **no** debes teclear las líneas 1 ni de la 9ØØØ a la 998Ø. Las puedes cargar al principio, o congregarlas (MERGE) posteriormente:

```
  1 LET cierre=9740:      LET cogebloque=9800: LET iniciar=9500:
    LET inreg=9000:       LET avipin=9140:     LET avipun=9300:
    LET exreg=9200:       LET dejabloque=9900: LET pinza=9600:
    LET arredro=9970:     LET prepare=9400:    LET punza=9700
  2 LET prexreg=8800:     LET sacaclave=7800:  LET cabosueltos=7600:
    LET compalit=7400:    LET companum=7200:   LET ordenar=7000:
    LET miraclavel=6800:  LET miraclaven=6600
 10 CLS
 20 PRINT AT 0,5;"Archivero Spectrum"
 25 GO SUB iniciar
 26 CLS
 30 PRINT AT 2,0;"Las opciones son:"
 40 PRINT AT 3,11;"1/ crear"
 41 PRINT AT 4,11;"2/ agregar"
 42 PRINT AT 5,11;"3/ suprimir"
 43 PRINT AT 6,11;"4/ escrutar"
 44 PRINT AT 7,11;"5/ acumular"
 45 PRINT AT 8,11;"6/ escapar"

100 INPUT "Teclee numero de opcion:";opt
110 GO SUB 200*opt
120 GO SUB arredro
130 GO TO 26

200 GO SUB inreg
210 GO SUB punza
220 INPUT "Otro registro?(s/n)";q$
230 IF q$="s" THEN  GO TO 200
235 CLS
240 GO SUB cierre
250 RETURN

400 INPUT "Cuantos registros?";nr
410 DIM b$(nr,lpr)
420 FOR q=1 TO nr
430 GO SUB inreg
440 LET b$(q)=r$
450 NEXT q
460 GO SUB sacaclave
465 GO SUB ordenar
470 LET ap=1
480 GO SUB pinza
490 IF ap>nr OR fifi THEN  GO SUB cabosueltos: RETURN
500 LET s$=r$(comienzo TO termino):
    LET t$=b$(p)(comienzo  TO termino)
510 IF ltipo THEN  GO SUB compalit
520 IF NOT ltipo THEN  GO SUB companum
530 IF comp<0 THEN  GO SUB punza: GO TO 480
540 IF comp>=0 THEN
    LET t$=r$: LET r$=b$(p): GO SUB punza:
    LET r$=t$: LET ap=ap+1: GO TO 490
```

```
 600 INPUT "Cuantos registros?";nr
 610 GO SUB sacaclave
 620 DIM d$(nr,termino-comienzo+1)
 630 FOR p=1 TO nr
 640 INPUT "Clave a suprimir(solo clave):";d$(p)
 650 NEXT p
 660 GO SUB pinza
 670 IF fifi THEN  GO SUB cierre: RETURN
 680 FOR p=1 TO nr
 690 IF r$(comienzo TO termino)=d$(p) THEN  GO TO 660
 700 NEXT p
 710 GO SUB punza
 720 GO TO 660

 800 GO SUB sacaclave
 810 IF ltipo THEN  DIM k$(termino-comienzo+1):
     INPUT "Clave de busqueda:";k$
 820 IF NOT ltipo THEN
     INPUT "Cota MIN:";cinf,"Cota MAX:";csup
 830 INPUT "Pantalla(1),Impresora(2)Cassette(3):";opt
 840 GO SUB pinza
 850 IF fifi AND opt=3 THEN  GO SUB cierre: RETURN
 860 IF fifi THEN  INPUT "Pulsa 'c' para continuar";k$: RETURN
 870 IF ltipo THEN  GO SUB miraclavel
 880 IF NOT ltipo THEN  GO SUB miraclaven
 890 IF NOT concuerda THEN  GO TO 840
 895 LET bs=comienzo: LET es=termino
 900 IF opt=1 THEN  GO SUB exreg
 910 IF opt=2 THEN  GO SUB prexreg
 920 IF opt=3 THEN  GO SUB punza
 925 LET comienzo=bs: LET termino=es
 930 GO TO 840

1000 LET sum=0
1010 GO SUB sacaclave
1020 IF ltipo THEN  PRINT "Clave no numerica": PAUSE 120: RETURN
1030 GO SUB pinza
1040 IF fifi THEN  PRINT "Suma de□";k$;"=□";sum:
     INPUT "Pulsa'c'para continuar";k$: RETURN
1050 LET sum=sum+VAL r$(comienzo TO termino)
1060 GO TO 1030

1200 STOP

6600 LET concuerda=1
6610 IF VAL r$(comienzo TO termino)<cinf OR
        VAL r$(comienzo TO termino)>csup THEN  LET concuerda=0
6620 RETURN

6800 LET concuerda=0
6810 IF k$=r$(comienzo TO termino) THEN  LET concuerda=1
6820 RETURN

7000 LET inc=1
7005 LET testigo=0
7010 FOR p=1 TO nr-inc
7020 LET s$=b$(p)(comienzo TO termino):
     LET t$=b$(p+1)(comienzo TO termino)
```




```
7030 IF ltipo THEN  GO SUB compalit
7040 IF NOT ltipo THEN  GO SUB companum
7050 IF comp>0 THEN  LET t$=b$(p):
     LET b$(p)=b$(p+1): LET b$(p+1)=t$: LET testigo=1
7060 NEXT p
7070 IF testigo>0 THEN  LET inc=inc+1: GO TO 7005
7080 RETURN

7200 IF VAL s$<VAL t$ THEN  LET comp=-1
7210 IF VAL s$=VAL t$ THEN  LET comp=0
7220 IF VAL s$>VAL t$ THEN  LET comp=1
7230 RETURN

7400 IF s$<t$ THEN  LET comp=-1
7410 IF s$=t$ THEN  LET comp=0
7420 IF s$>t$ THEN  LET comp=1
7430 RETURN

7600 IF ap>nr THEN  GO TO 7670
7610 FOR p=ap TO nr
7620 LET r$=b$(p)
7630 GO SUB punza
7640 NEXT p
7650 GO SUB cierre
7660 RETURN
7670 GO SUB punza
7680 GO SUB pinza
7690 IF fifi THEN  GO SUB cierre: RETURN
7700 GO TO 7670

7800 DIM k$(9): INPUT "Teclee nombre del campo clave   ";k$
7810 LET comienzo=1
7820 FOR p=2 TO 11
7830 IF n$(p)(2 TO )=k$ THEN  GO TO 7860
7840 LET comienzo=comienzo+w(p)
7850 NEXT p
7855 PRINT "LA CLAVE TECLEADA NO EXISTE ": GO TO 7800
7860 LET termino=comienzo+w(p)-1
7870 IF n$(p,1)="l" THEN  LET ltipo=1
7880 IF n$(p,1)="n" THEN  LET ltipo=0
7890 RETURN

8800 LET comienzo=1
8810 FOR p=2 TO 11
8815 IF n$(p,1)="□" THEN  LPRINT : RETURN
8820 LPRINT n$(p,1);":";TAB 4;n$(p)(2 TO );
8840 LET termino=comienzo+w(p)-1
8850 LPRINT TAB 15;r$(comienzo TO termino)
8860 LET comienzo=termino+1
8870 NEXT p
8880 LPRINT : RETURN

9000 DIM a$(lpr)
9010 CLS : LET comienzo=1
9020 FOR p=2 TO 11
9025 IF n$(p,1)="□" THEN  GO TO 9120
9030 PRINT AT p,0;n$(p,1);":";AT p,4;n$(p)(2 TO )
9040 FOR c=1 TO w(p)
9050 PRINT AT p,14+c;"_"
          c
```



```
9070 LET termino=comienzo+w(p)-1
9080 INPUT (n$(p)(2 TO ));a$(comienzo TO termino)
9090 PRINT AT p,15;a$(comienzo TO termino)
9100 LET comienzo=termino+1
9110 NEXT p
9120 LET r$=a$
9130 RETURN

9140 PRINT  INVERSE 1;"Marche" AND NOT testigo;
          "Quiete" AND testigo;" Rascando"
9150 BEEP .2,15: PAUSE 6: BEEP .3,20: PAUSE 80
9160 LET testigo=NOT testigo
9170 RETURN

9200 LET comienzo=1
9210 FOR p=2 TO 11
9215 IF n$(p,1)="□" THEN  PRINT : RETURN
9220 PRINT n$(p,1);":";TAB 4;n$(p)(2 TO );
9240 LET termino=comienzo+w(p)-1
9250 PRINT TAB 15;r$(comienzo TO termino)
9260 LET comienzo=termino+1
9270 NEXT p
9280 PRINT : RETURN

9300 PRINT  INVERSE 1;"Marche" AND NOT testigo;
          "Quiete" AND testigo;" Grabando"
9310 BEEP .2,20: PAUSE 6: BEEP .3,5: PAUSE 80
9320 LET testigo=NOT testigo
9330 RETURN

9400 INPUT "Numero de campos: ";nf
9405 CLS
9410 FOR p=2 TO nf+1
9415 PRINT AT 10,2;"Campo□";p-1
9420 INPUT "Nombre del campo  (precedido de l/n):";n$(p)
9422 IF n$(p,1)<>"l" AND n$(p,1)<>"n" THEN  GO TO 9420
9425 INPUT "Numero de octetos: ";w(p)
9430 NEXT p
9435 CLS
9440 INPUT "No. de registros por bloque: ";w(1)
9445 RETURN

9500 DIM n$(11,10): DIM w(11): LET lpr=0: GO SUB arredro
9505 INPUT "Nombre fichero de entrada:      ";f$
9510 IF f$="nulo" THEN  GO SUB prepare: GO TO 9525
9514 GO SUB avipin
9515 LOAD f$+"h1" DATA n$()
9520 LOAD f$+"h2" DATA w()
9521 GO SUB avipin
9525 INPUT "Nombre fichero de salida:       ";g$
9530 FOR p=2 TO 11
9535 LET lpr=lpr+w(p)
9540 NEXT p
9545 DIM i$(w(1),lpr): DIM e$(w(1),lpr)
9547 IF g$="nulo" THEN  RETURN
9548 GO SUB avipun
9550 SAVE g$+"h1" DATA n$()
9555 SAVE g$+"h2" DATA w()
9557 GO SUB avipun
9560 RETURN
```




```
9600 IF pin=0 OR pin>w(1) THEN  GO SUB cogebloque
9610 IF i$(pin)( TO 7)="ultimal" THEN
     PRINT "Intento de sobrepasar fifi": STOP
9620 LET r$=i$(pin)
9625 IF r$( TO 2)="çç" THEN  LET fifi=1
9630 LET pin=pin+1
9640 RETURN

9700 LET pex=pex+1
9710 LET e$(pex)=r$
9720 IF pex=w(1) OR r$="ultimal" THEN  GO SUB dejabloque
9730 RETURN

9740 LET r$="çç":
9750 GO SUB punza
9760 LET r$="ultimal"
9770 GO SUB punza
9780 RETURN

9800 LET m$=STR$ inbc
9810 GO SUB avipin
9820 LOAD f$+m$ DATA i$()
9825 POKE 23692,255
9830 GO SUB avipin
9840 LET pin=1
9850 LET inbc=inbc+1
9860 RETURN

9900 LET m$=STR$ exbc
9910 GO SUB avipun
9920 SAVE g$+m$ DATA e$()
9930 GO SUB avipun
9940 LET pex=0
9950 LET exbc=exbc+1
9960 RETURN

9970 LET pin=0: LET pex=0: LET inbc=0:
     LET exbc=0: LET fifi=0: LET testigo=0
9980 RETURN
```



APENDICE E: Construyendo tu propio Conmutador Load/Save

¿Cuántas veces has olvidado quitar la clavija "ear" al guardar un programa en cinta? En la quincuagésima ocasión en que me ocurrió, decidí que ya era hora de adoptar medidas para impedirlo. Aquí hay una solución práctica y simple.

Necesitas:

(a) clavijas cilíndricas 4 x 3.5 mm.
(b) un conmutador miniatura deslizante de 2 polos 2 vías
(c) 2 metros de cable microfónico apantallado (de una sola alma).
(d) Una vieja cajita de plástico de cassette.

(a), (b), y (c) están disponibles en cualquier tienda electrónica por un valor aproximado de 200 pesetas cada una, y (d) puedes aprovechar cualquier cajetín de plástico.

Taladra cuatro agujeros en los lados estrechos de la caja, lo suficientemente grandes como para permitir pasar los cables a través de ellos. Haz otro agujero en una de las caras grandes para montar el conmutador deslizante. (La forma más fácil de que el agujero sea del tamaño adecuado, es cortar un trozo de cinta adhesiva que actúe como plantilla. Ahora haz las conexiones que aparecen en la figura E1.

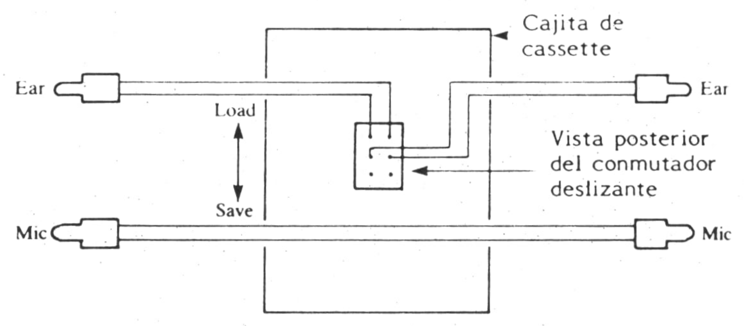


Figura E.1 Diagrama del circuito para conmutador LOAD/SAVE.

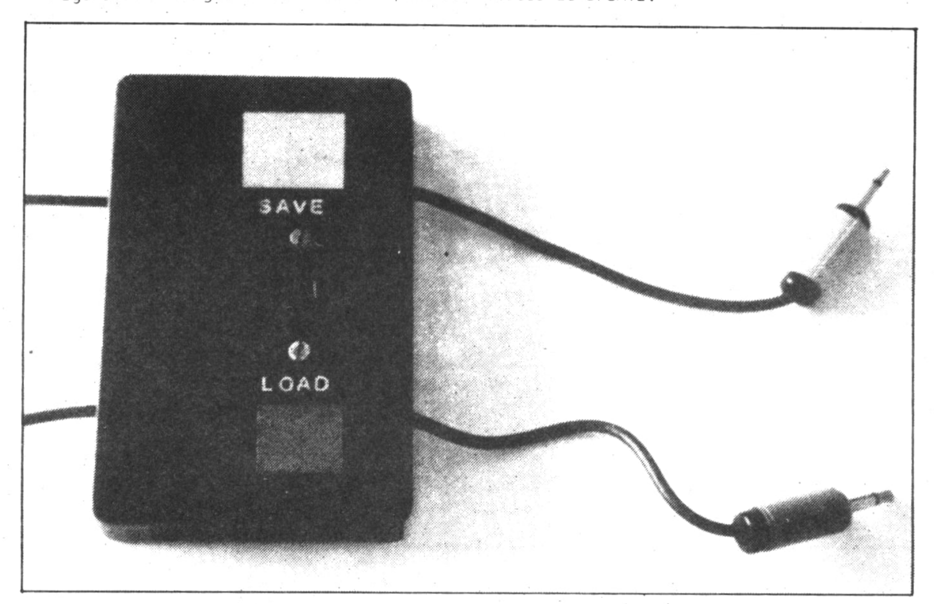


Figura E.2 Vista frontal del conmutador.



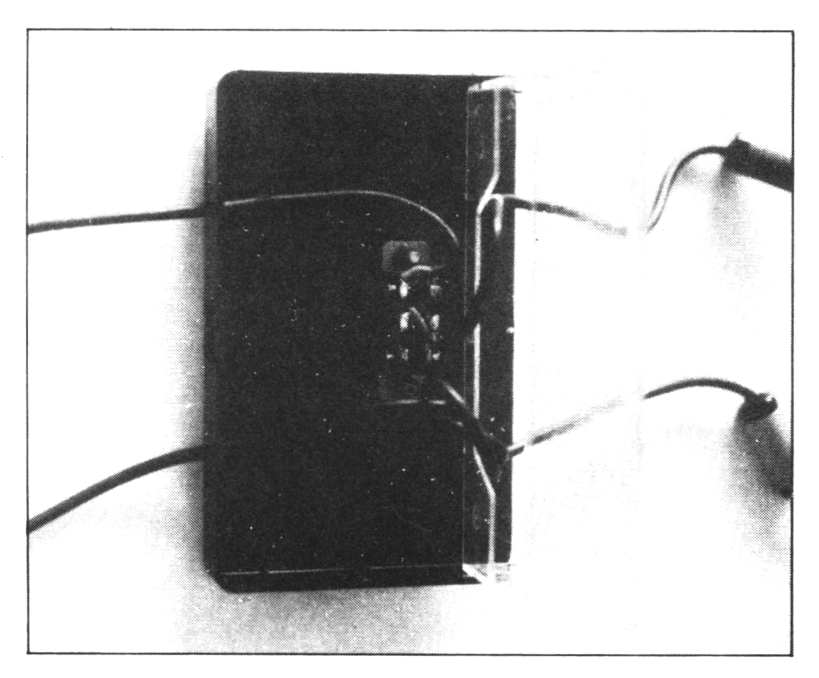


Figura E.3 Vista trasera del conmutador

Finalmente, marca de colores las clavijas con cinta adhesiva para evitar confusión.

Todo esto lo que hace, es interrumpir el cable "ear" de manera que cuando guardas en cinta colocas el conmutador en la posición SAVE (lo lógico); y lo deslizas de nuevo hacia la posición LOAD para cargar o verificar lo grabado. Observa que el cable "mic" no está realmente conectado a nada dentro de la caja, pero es conveniente que la atraviese simplemente para evitar perderlo.




PROGRAMACION AVANZADA ZX SPECTRUM